**Fake News Predictor**

Summer Internship Report

submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology

in

Information Technology

By

**Ashish Dagar – 00696303116**

****

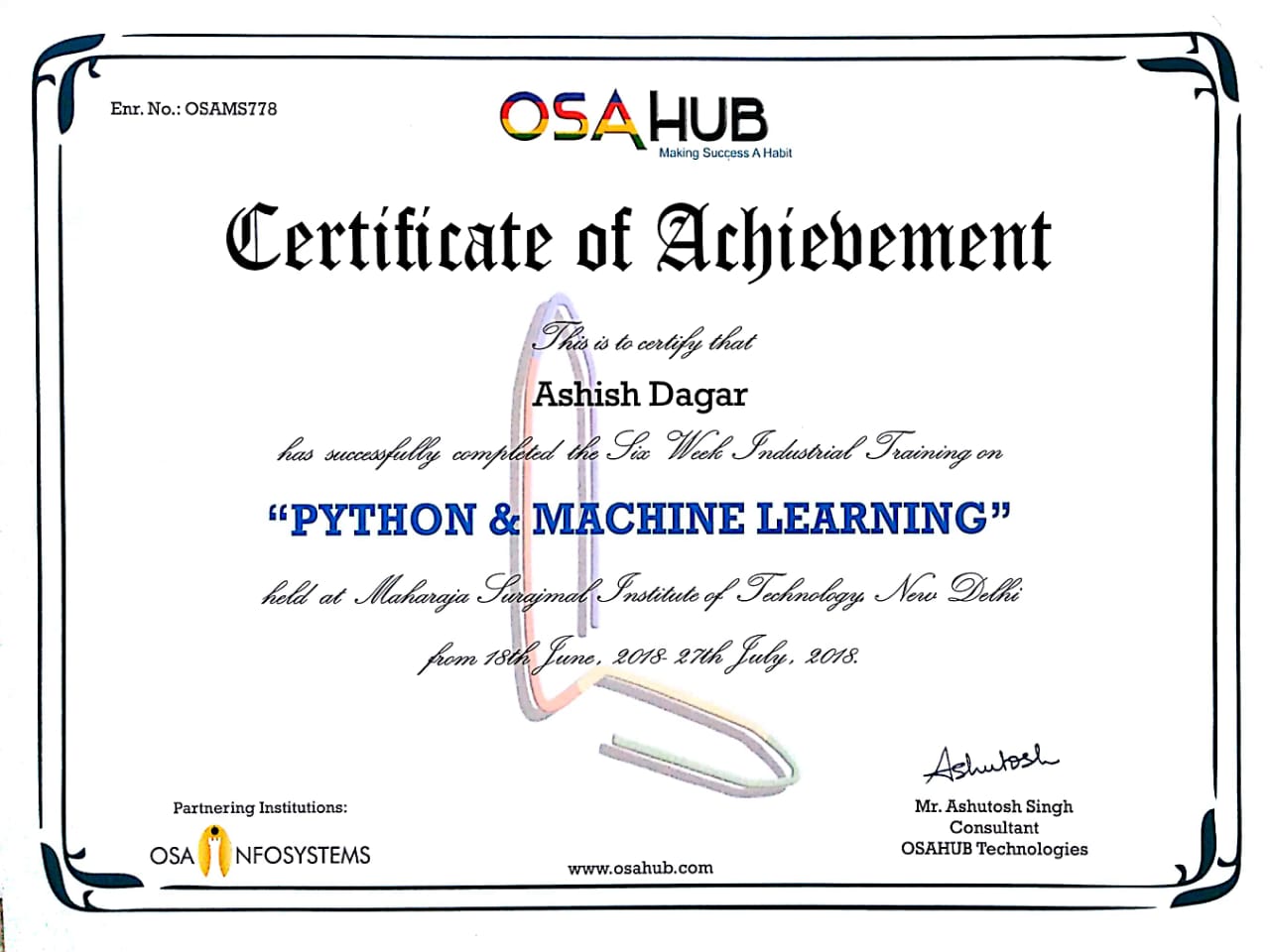
Maharaja Surajmal Insitute of Technology

(Affiliated to Guru Gobind Singh Indraprastha University)

Janakpuri, New Delhi-58

October 2018

**Cretificate**

****

**Acknowledgement**

A project work owes its success from commencement to completion to the people in love with project at various stages. Let me in this page express my gratitude to all those who helped in various stages. First I would like to express my sincere gratitude indebtness to **Mr. Manoj Malik** (HOD, Department of Information Technology) & **Mr. Satender Malik** (Proctor of IT Evening Shift) for allowing me to undergo the summer training of 6 weeks at OSAHUB, MSIT.

I am grateful to our guide **Mr. Ashutosh Singh** for the help provided in completion of the project, which was assigned to me. Whithout his friendly help and guidance it was difficult to develop this project.

Last but not least, I pay my sincere thanks and gratitude to all the staff members of **OSAHUB** for their support and for making our training valuable and fruitful.

**Table of Content**

**Title Page………………………………………………………… 1**

**Certificate………………………………………………………... 2**

**Acknowledgment………………………………………………… 3**

**Abstract of Project……………………………………………… 5**

**Abstract of Project**

**Chapter 1 : Company Profile**

**C:\Users\himalaya\Downloads\osahubLogo.png**

OSAHUB Technologies is India's leading training provider. We focus on education solutions, covering all the latest technologies - be it in the field of **Computer Science, Electronics, Mechanical Engineering, Robotics, Animation, etc.** We are Ranked among Top Training Institutions / Education Centers and Graded among India's Most Trusted Service Brands. We provide Workshops, Trainings, Faculty Development Programs, Certifications Courses, Seminars, Career Counseling, etc. to engineering colleges and schools all across India. We have an extremely skilled network of subject matter experts and highly experienced trainers who have several years of experience in the industry. We are pioneers in the education industry and are proud of our excellent team.

Our vision is to act as a mediator between Technology and Students to improve their academic performance by providing non-syllabus inputs and Best Trainings on various Emerging Technologies. Our vision is to empower youth through high quality and dedicated education.

**Chapter 2: Technology Used**

**2.1 PYTHON**

 Python is one of those rare languages which can claim to be both **simple** and **powerful**. You will be pleasantly surprised to see how easy it is to concentrate on the solution to the problem rather than on the syntax (i.e. the structure of the program that you are writing) of the language.

The official Python introduction is

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

I will discuss these features in more detail in the next section.

By the way, Guido van Rossum (the creator of the Python language) named the language after the BBC show "Monty Python's Flying Circus". He doesn't particularly like snakes that kill animals for food by winding their long bodies around them and crushing them.

**Features of Python**

**Simple**

Python is a simple and minimalistic language. Reading a good Python program feels almost like reading English (but very strict English!). This pseudo-code nature of Python is one of its greatest strengths. It allows you to concentrate on the solution to the problem rather than the syntax i.e. the language itself.

**Easy to Learn**

As you will see, Python is extremely easy to get started with. Python has an extraordinarily simple syntax as already mentioned.

**Free and Open Source**

Python is an example of a FLOSS (Free/Libre and Open Source Software). In simple terms, you can freely distribute copies of this software, read the software's source code, make changes to it, use pieces of it in new free programs, and that you know you can do these things. FLOSS is based on the concept of a community which shares knowledge. This is one of the reasons why Python is so good - it has been created and improved by a community who just want to see a better Python.

**High-level Language**

When you write programs in Python, you never need to bother about low-level details such as managing the memory used by your program.

**Portable**

Due to its open-source nature, Python has been ported (i.e. changed to make it work on) to many many platforms. All your Python programs will work on any of these platforms without requiring any changes at all. However, you must be careful enough to avoid any system-dependent features.

You can use Python on Linux, Windows, Macintosh, Solaris, OS/2, Amiga, AROS, AS/400, BeOS, OS/390, z/OS, Palm OS, QNX, VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE and PocketPC !

**Interpreted**

This requires a little explanation.

A program written in a compiled language like C or C++ is translated from the source language i.e. C/C++ into a language spoken by your computer (binary code i.e. 0s and 1s) using a compiler with various flags and options. When you run the program, the linker/loader software just stores the binary code in the computer's memory and starts executing from the first instruction in the program.

When you use an interpreted language like Python, there is no separate compilation and execution steps. You just *run* the program from the source code. Internally, Python converts the source code into an intermediate form called bytecodes and then translates this into the native language of your specific computer and then runs it. All this makes using Python so much easier. You just *run* your programs - you never have to worry about linking and loading with libraries, etc. They are also more portable this way because you can just copy your Python program into another system of any kind and it just works!

**Object Oriented**

Python supports procedure-oriented programming as well as object-oriented programming. In *procedure-oriented* languages, the program is built around procedures or functions which are nothing but reusable pieces of programs. In *object-oriented* languages, the program is built around objects which combine data and functionality. Python has a very powerful but simple way of doing object-oriented programming, especially, when compared to languages like C++ or Java.

**Extensible**

If you need a critical piece of code to run very fast, you can achieve this by writing that piece of code in C, and then combine that with your Python program.

**Embeddable**

You can embed Python within your C/C++ program to give scripting capabilities for your program's users.

**Extensive Libraries**

The Python Standard Library is huge indeed. It can help you do various things involving regular expressions, documentation generation, unit testing, threading, databases, web browsers, CGI, ftp, email, XML, XML-RPC, HTML, WAV files, cryptography, GUI(graphical user interfaces) using Tk, and also other system-dependent stuff. Remember, all this is always available wherever Python is installed. This is called the "batteries included" philosophy of Python.

Besides the standard library, there are various other high-quality libraries such as the [Python Imaging Library](http://www.pythonware.org/) which is an amazingly simple image manipulation library.

**2.2 JUPYTER NOTEBOOK**

 In this case, "notebook" or "notebook documents" denote documents that contain both code and rich text elements, such as figures, links, equations, ... Because of the mix of code and text elements, these documents are the ideal place to bring together an analysis description and its results as well as they can be executed perform the data analysis in real time.

These documents are produced by the Jupyter Notebook App.

For now, you should just know that "Jupyter" is a loose acronym meaning Julia, Python, and R. These programming languages were the first target languages of the Jupyter application, but nowadays, the notebook technology also supports [many other languages](http://github.com/ipython/ipython/wiki/IPython-kernels-for-other-languages).

As you just saw, the main components of the whole environment are, on the one hand, the notebooks themselves and the application. On the other hand, you also have a notebook kernel and a notebook dashboard.

Let's look at these components in more detail.

### 2.2.1 Jupyter Notebook App

As a server-client application, the Jupyter Notebook App allows you to edit and run your notebooks via a web browser. The application can be executed on a PC without Internet access or it can be installed on a remote server, where you can access it through the Internet.

Its two main components are the kernels and a dashboard.

A kernel is a program that runs and introspects the user’s code. The Jupyter Notebook App has a kernel for Python code, but there are also kernels available for other programming languages.

The dashboard of the application not only shows you the notebook documents that you have made and can reopen but can also be used to manage the kernels: you can which ones are running and shut them down if necessary.

## 2.2.1 History of IPython and Jupyter Notebooks

To fully understand what the Jupyter Notebook is and what functionality it has to offer you need to know how it originated.

Let's back up briefly to the late 1980s. Guido Van Rossum begins to work on Python at the National Research Institute for Mathematics and Computer Science in the Netherlands..

Let's go to late 2001, twenty years later. Fernando Pérez starts developing IPython.

In 2005, both Robert Kern and Fernando Pérez attempted building a notebook system. Unfortunately, the prototype had never become fully usable.

Fast forward two years: the IPython team had kept on working, and in 2007, they formulated another attempt at implementing a notebook-type system. By October 2010, there was a prototype of a web notebook and in the summer of 2011, this prototype was incorporated and it was released with 0.12 on December 21, 2011. In subsequent years, the team got awards, such as the Advancement of Free Software for Fernando Pérez on 23 of March 2013 and the Jolt Productivity Award, and funding from the Alfred P. Sloan Foundations, among others.

Lastly, in 2014, Project Jupyter started as a spin-off project from IPython. IPython is now the name of the Python backend, which is also known as the kernel. Recently, the next generation of Jupyter Notebooks has been introduced to the community. It's called JupyterLab. Read more about it [here](http://blog.jupyter.org/2016/07/14/jupyter-lab-alpha/). After all this, you might wonder where this idea of notebooks originated or how it came about to the creators.

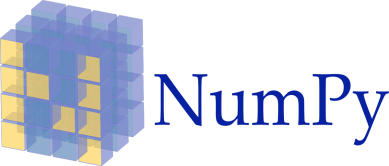
A brief research into the history of these notebooks learns that Fernando Pérez and Robert Kern were working on a notebook just at the same time as the Sage notebook was a work in progress. Since the layout of the Sage notebook was based on the layout of Google notebooks, you can also conclude that also Google used to have a notebook feature around that time.

For what concerns the idea of the notebook, it seems that Fernando Pérez, as well as William Stein, one of the creators of the Sage notebook, have confirmed that they were avid users of the Mathematica notebooks and Maple worksheets. The Mathematica notebooks were created as a front end or GUI in 1988 by Theodore Gray.

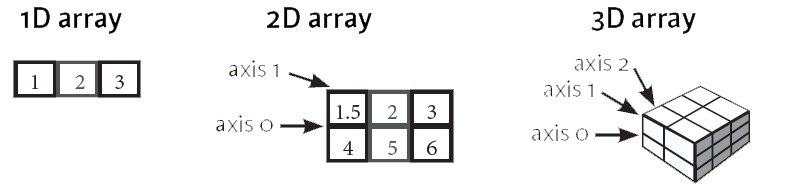
The concept of a notebook, which contains ordinary text and calculation and/or graphics, was definitely not new.

Also, the developers had close contact with one another and this, together with other failed attempts at GUIs for IPython and the use of "AJAX" = web applications, which didn't require users to refresh the whole page every time you do something, were two other motivations for the team of William Stein to start developing the Sage notebooks.

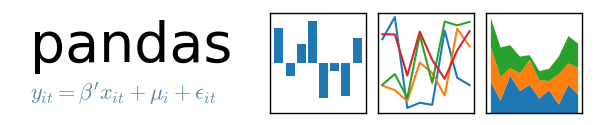
**2.3 NUMPY**

 NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed. This tutorial explains the basics of NumPy such as its architecture and environment. It also discusses the various array functions, types of indexing, etc. An introduction to Matplotlib is also provided. All this is explained with the help of examples for better understanding.

**Numeric**, the ancestor of NumPy, was developed by Jim Hugunin. Another package Numarray was also developed, having some additional functionalities. In 2005, Travis Oliphant created NumPy package by incorporating the features of Numarray into Numeric package. There are many contributors to this open source project.



**2.4 PANDAS**

 Pandas is a library that unifies the most common workflows that data analysts and data scientists previously relied on many different libraries for. Pandas has quickly became an important tool in a data professional's toolbelt and is the most popular library for working with tabular data in Python. Tabular data is any data that can be represented as rows and columns. The CSV files we've worked with in previous missions are all examples of tabular data.

To represent tabular data, pandas uses a custom data structure called a **dataframe**. A dataframe is a highly efficient, 2-dimensional data structure that provides a suite of methods and attributes to quickly explore, analyze, and visualize data. The dataframe is similar to the NumPy 2D array but adds support for many features that help you work with tabular data.

One of the biggest advantages that pandas has over NumPy is the ability to store mixed data types in rows and columns. Many tabular datasets contain a range of data types and pandas dataframes handle mixed data types effortlessly while NumPy doesn't. Pandas dataframes can also handle missing values gracefully using a custom object, NaN, to represent those values. A common complaint with NumPy is its lack of an object to represent missing values and people end up having to find and replace these values manually. In addition, pandas dataframes contain axis labels for both rows and columns and enable you to refer to elements in the dataframe more intuitively. Since many tabular datasets contain column titles, this means that dataframes preserve the metadata from the file around the data.

**2.5 SciKit-Learn**

 Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python.

It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use.

The library is built upon the SciPy (Scientific Python) that must be installed before you can use scikit-learn. This stack that includes:

* **NumPy**: Base n-dimensional array package
* **SciPy**: Fundamental library for scientific computing
* **Matplotlib**: Comprehensive 2D/3D plotting
* **IPython**: Enhanced interactive console
* **Sympy**: Symbolic mathematics
* **Pandas**: Data structures and analysis

Extensions or modules for SciPy care conventionally named [SciKits](http://scikits.appspot.com/scikits). As such, the module provides learning algorithms and is named scikit-learn.

The vision for the library is a level of robustness and support required for use in production systems. This means a deep focus on concerns such as easy of use, code quality, collaboration, documentation and performance.

Although the interface is Python, c-libraries are leverage for performance such as numpy for arrays and matrix operations, [LAPACK](http://www.netlib.org/lapack/), [LibSVM](http://www.csie.ntu.edu.tw/~cjlin/libsvm/) and the careful use of cython.

Take my free 2-week email course and discover data prep, algorithms and more (with code).

The library is focused on modeling data. It is not focused on loading, manipulating and summarizing data. For these features, refer to NumPy and Pandas.

Screenshot taken from[a demo of the mean-shift clustering algorithm](http://scikit-learn.org/stable/auto_examples/cluster/plot_mean_shift.html)

Some popular groups of models provided by scikit-learn include:

* **Clustering**: for grouping unlabeled data such as KMeans.
* **Cross Validation**: for estimating the performance of supervised models on unseen data.
* **Datasets**: for test datasets and for generating datasets with specific properties for investigating model behavior.
* **Dimensionality Reduction**: for reducing the number of attributes in data for summarization, visualization and feature selection such as Principal component analysis.
* **Ensemble methods**: for combining the predictions of multiple supervised models.
* **Feature extraction**: for defining attributes in image and text data.
* **Feature selection**: for identifying meaningful attributes from which to create supervised models.
* **Parameter Tuning**: for getting the most out of supervised models.
* **Manifold Learning**: For summarizing and depicting complex multi-dimensional data.
* **Supervised Models**: a vast array not limited to generalized linear models, discriminate analysis, naive bayes, lazy methods, neural networks, support vector machines and decision trees.