A MAJOR PROJECT REPORT

ON

SMART RESUME ANALYZER

Submitted by

Sampath Kumar Kolichalam (19CS38)

Under the Guidance of

Ms. Komal Malsa

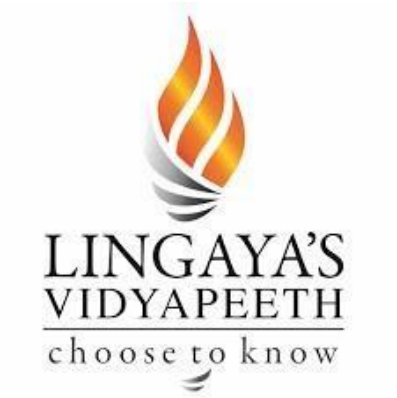
Assistant Professor

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING



Department of Computer Science & Engineering

LINGAYAS VIDYAPEETH, FARIDABAD

May-June, 2023

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our supervisor **Ms. Komal Malsa, Assistant Professor, Department of Computer Science and Engineering,** for giving us the opportunity to work on this topic. It would never be possible for us to take this project to this level without their innovative ideas and their relentless support and encouragement.

I express my deepest thanks to **other faculties who have contributed** for taking part in useful decision & giving necessary advice and guidance and arranged all facilities to make life easier. I choose this moment to acknowledge his/her contribution gratefully.

We take immense pleasure in thanking **Dr. Bindiya Ahuja**, Head of the School of Computer Science and Information Technology (Specialization). Her tolerance of having discussions, sacrificing her personal time, are extremely insightful and greatly appreciated.

Sampath Kumar Kolichalam, 19CS38

DECLARATION

We hereby declare that this project report entitled "**SMART RESUME ANALYZER**" by **SAMPATH KUMAR KOLICHALAM(19CS38),** being submitted in partial fulfillment of the requirements for the degree of Bachelor of Technology in **Computer Science & Engineering** under the Department of Computer Science and Engineering of Lingayas Vidyapeeth Faridabad, during the academic year2022-2023, is a Bonafede record of our original work carried out under the guidance of **MS. Komal Malsa, Assistant Professor, Department of Computer Science and Engineering.**

We further declare that we have not submitted the matter presented in this Project for the award of any other Degree/Diploma of this University or any other University/Institute.

Sampath Kumar Kolichalam (19CS38)

Lingayas Vidyapeeth, Faridabad

Department of Computer Science & Engineering

May-June, 2023

CERTIFICATE

This is to certify that this project report entitled **SMART RESUME ANALYZER** by **Sampath Kumar Kolichalam (19CS38),** submitted in partial fulfillment of the requirements for the degree of Bachelor of Technology in **Computer Science and Engineering** under the Department of Computer Science and Engineering, Lingayas Vidyapeeth, Faridabad, during the academic year 2022-23, is a Bonafide record of work carried out under my guidance and supervision.

(Signature of Project Guide)

Ms. Komal Malsa
Assistant Professor
School of Computer Science & Information Technology
Lingayas Vidyapeeth, Faridabad

Dr. Bindiya Ahuja
Professor & Head of Department
School of Computer Science & Information Technology
Lingayas Vidyapeeth, Faridabad

TABLE OF CONTENTS

List Of Figures

List Of Screenshots

Abstract **1**

Chapter-1 Introduction **2-7**

1.1 Introduction

1.2 Existing System

1.3 Proposed System

1.4 Working

1.5 Overview

Chapter-2 Literature Survey **8**

Chapter-3 NLP **9-12**

3.1 NLP

3.2 What Are the Goals of NLP

3.3 Applications of NLP

3.4 Subsets of NLP

3.5 Types of NLP

Chapter-4 Software Requirements & Algorithms **13-20**

4.1 Introduction to Python

4.2 History of Python

4.3 Features of Python

4.4 Downloading of Python

4.5 Installing of Python

4.6 Python Libraries

4.7 Project Creation Steps

Chapter-5 System Design & Testing **21-36**

5.1 Introduction

5.2 Algorithm & Implementation

5.3 Algorithm Model Flow Diagram
5.4 Sample Code
5.5 NLP Testing

Chapter-6 Output Screens **37-38**

Chapter-7 Conclusion **39**

Chapter-8 Bibliography **40**

LIST OF FIGURES

Fig No	Name	Page No
Fig 5.2	Algorithm & implementation	22
Fig 5.3	Algorithm model and flow diagram	23
Fig 6.1	Project Interface	37
Fig 6.2	Resume Data	37
Fig 6.3	Resume Uploading	38
Fig 6.4	Users Resumes	38

LIST OF SCREENSHOTS

Fig No	Name	Page No
4.1	Pycharm Tool	17
4.2	Pycharm Projects	18
4.3	Pycharm Files	18
4.4	Pycharm Myproject	19
4.5	Pycharm Tool Activites	19
4.6	Python Files	20
4.7	Multiple Python Files	20

ABSTRACT

The Smart Resume Analyzer is an intelligent tool designed to automate the process of analyzing and evaluating resumes. It leverages natural language processing (NLP) and machine learning techniques to extract relevant information from resumes and provide meaningful insights to recruiters and hiring managers.

Traditional resume screening methods can be time-consuming and prone to human bias. The Smart Resume Analyzer aims to streamline this process by automatically parsing resumes, identifying key details, and assessing candidates based on predefined criteria. By utilizing advanced NLP algorithms, it can accurately extract information such as contact details, educational background, work experience, skills, and achievements from resumes in various formats.

The analyzer employs machine learning models to evaluate resumes based on customizable criteria. Recruiters can define specific requirements, such as educational qualifications, years of experience, and preferred skills, to rank and filter candidates effectively. The system can also be trained on historical data to learn from past hiring decisions and improve its accuracy over time.

The Smart Resume Analyzer offers several advantages over manual resume screening. It significantly reduces the time and effort required to review a large number of resumes, allowing recruiters to focus on more strategic aspects of the hiring process. Moreover, by automating the initial screening, it helps eliminate unconscious bias and promotes fairer candidate evaluations.

Additionally, the analyzer can generate comprehensive reports and visualizations, summarizing the analysis results and providing an overview of the candidate pool. This enables recruiters to make informed decisions and identify the most suitable candidates for further evaluation and interview stages.

In conclusion, the Smart Resume Analyzer combines NLP, machine learning, and automation to enhance the efficiency and objectivity of resume screening. By leveraging advanced technologies, it empowers recruiters to make data-driven hiring decisions, resulting in improved recruitment outcomes and a more streamlined hiring process.

CHAPTER-1
INTRODUCTION

1.1 INTRODUCTION

The Smart Resume Analyzer is a revolutionary tool designed to transform the traditional process of resume screening and analysis. In today's competitive job market, recruiters and hiring managers often face the challenge of reviewing numerous resumes to identify the most qualified candidates. This task can be time-consuming, subject to human bias, and prone to errors.

The Smart Resume Analyzer addresses these challenges by harnessing the power of advanced technologies such as natural language processing (NLP) and machine learning. It automates the resume screening process, allowing recruiters to efficiently analyze and evaluate a large number of resumes in a fraction of the time it would take manually.

With its intelligent algorithms, the analyzer can accurately extract key information from resumes, including contact details, education history, work experience, skills, and achievements. By understanding the context and meaning behind the text, it can identify relevant qualifications and match them against predefined criteria set by recruiters.

One of the significant advantages of the Smart Resume Analyzer is its ability to minimize bias in the screening process. Human recruiters may unintentionally be influenced by factors like a candidate's name, gender, or background. By relying on objective data analysis, the analyzer helps ensure fair and unbiased evaluations, providing equal opportunities for all candidates.

Moreover, the analyzer can be customized to align with specific job requirements and organizational preferences. Recruiters can define their criteria for evaluating candidates, such as minimum educational qualifications, years of experience, and desired skills. The analyzer then ranks and filters resumes based on these criteria, allowing recruiters to focus their attention on the most promising candidates.

Another key feature of the Smart Resume Analyzer is its learning capability. By leveraging machine learning algorithms, the analyzer can continuously improve its performance over time. It can be trained on historical data, incorporating insights

from successful past hiring decisions. This iterative learning process enhances the analyzer's accuracy and effectiveness in identifying the right candidates.

Additionally, the analyzer generates detailed reports and visualizations, summarizing the analysis results and providing an overview of the candidate pool. These insights help recruiters make informed decisions and select candidates for further evaluation and interviews.

Overall, the Smart Resume Analyzer revolutionizes the resume screening process by automating the analysis and evaluation of resumes. By leveraging advanced technologies, it saves time, reduces bias, and improves the efficiency of recruitment efforts. With its customizable criteria and learning capabilities, it empowers recruiters to make data-driven decisions, resulting in better hiring outcomes and a more streamlined hiring process.

1.2 Existing System

The Smart Resume Analyzer is a system designed to automate the process of analyzing and evaluating resumes submitted by job applicants. It utilizes various technologies, such as natural language processing (NLP) and machine learning, to extract relevant information from resumes and provide insights to employers or recruiters.

Existing systems for resume analysis typically follow a similar workflow:

Resume Parsing: The system extracts textual data from resumes, including personal information, educational background, work experience, skills, and achievements. This process may involve optical character recognition (OCR) to convert scanned or PDF resumes into machine-readable text.

1.3 Proposed system

The Smart Resume Analyzer is a proposed system designed to analyze resumes in an intelligent and efficient manner. It utilizes advanced technologies such as natural language processing (NLP) and machine learning to extract relevant information from resumes and provide valuable insights to recruiters or hiring managers. The system aims to streamline the resume screening process and enhance the overall recruitment experience.

1.4 Working

The Smart Resume Analyzer is an advanced tool designed to analyze and evaluate resumes or CVs (curriculum vitae). While I don't have access to specific information about a particular Smart Resume Analyzer, I can provide you with a general idea of how such a tool might work based on the technology and techniques commonly used in resume analysis.

Parsing and Extraction: The tool first parses the resume document to extract relevant information such as contact details, educational qualifications, work experience, skills, and achievements. This step involves employing natural language processing (NLP) techniques to understand the structure and content of the document.

Keyword Matching: The tool compares the extracted information with a predefined set of keywords or key phrases that are relevant to the job or industry. It checks if the resume includes the necessary skills, qualifications, and experiences that the employer is seeking.

Skill Assessment: The analyzer may utilize machine learning algorithms to evaluate the proficiency levels of specific skills mentioned in the resume. This assessment can be based on the frequency of usage, context, or even external data sources that indicate the applicant's expertise in a particular area.

HARDWARE REQUIREMENTS:

- System : Intel I3 2.4 GHz. Or Advanced
- Hard Disk : 200 GB +
- Monitor : 14’ Colour Monitor. Or Advanced ✞ Mouse : Optical Mouse.
- Ram : 4GB +
- GRAPHICS CARD: Integrated Graphics

SOFTWARE REQUIREMENTS:

- Operating system : Windows / Linux ✞ Coding Language : Python / anaconda
- Front-End : Python.

1.5 OVERVIEW

The Smart Resume Analyzer is an advanced tool designed to analyze and evaluate resumes in a smart and efficient manner. It utilizes artificial intelligence and natural language processing techniques to extract relevant information from resumes and provide valuable insights to both job seekers and recruiters.

Key Features:

Resume Parsing: The Smart Resume Analyzer automatically extracts important details such as contact information, work experience, education, skills, and achievements from resumes.

It saves time and effort by eliminating the need for manual data entry.

Keyword Optimization: The tool analyzes the job description or specific keywords provided by the user and matches them with the content of the resume. It highlights the presence or absence of relevant keywords, helping job seekers tailor their resumes to specific job requirements.

Skills Assessment: The Smart Resume Analyzer identifies and assesses the skills mentioned in the resume, providing an overview of the candidate's proficiency in different areas. This feature helps job seekers understand which skills are emphasized and allows recruiters to quickly assess the suitability of applicants.

Language and Grammar Check: The Smart Resume Analyzer checks the resume for language proficiency, grammar errors, and spelling mistakes. It provides suggestions for improvements, helping candidates create error-free and professional resumes.

Resume Score and Feedback: Based on the analysis of various resume components, the tool generates a score or rating for the resume. It provides detailed feedback on areas that need improvement, such as formatting, content, or presentation. This feedback helps candidates enhance their resumes and increase their chances of getting noticed by recruiters.

Benefits:

Time-saving: The Smart Resume Analyzer automates the resume review process, reducing the time and effort required to evaluate resumes manually.

Customization: Job seekers can tailor their resumes to specific job requirements by optimizing keywords and highlighting relevant skills.

Increased Visibility: By optimizing the resume for ATS compatibility, the tool improves the chances of the resume being selected for further consideration by recruiters.

Professional Presentation: The language and grammar check feature ensures that the resume is error-free, enhancing the professional presentation and impression on potential employers.

Self-improvement: The detailed feedback and suggestions provided by the tool enable candidates to enhance their resumes and improve their overall job application strategy.

The Smart Resume Analyzer is a valuable tool for both job seekers and recruiters, streamlining the resume evaluation process and increasing the efficiency of the job application process.

Pre-Processing:

Pre-processing is an important step in any data analysis or machine learning task, including the analysis of resumes using a Smart Resume Analyzer. The goal of pre-processing is to clean and prepare the resume data so that it can be effectively analyzed and interpreted by the system. Here are some common pre-processing steps for a Smart Resume Analyzer:

Data Extraction: The first step is to extract the relevant information from the resume documents. This may involve parsing the resume files in various formats such as PDF, Word, or plain text. The extracted data typically includes sections like contact information, education history, work experience, skills, and other relevant details.

Text Cleaning: The extracted text may contain noise or irrelevant information that needs to be removed. Common cleaning steps include removing special characters, punctuation marks, and HTML tags (if any). Additionally, the text may be converted to lowercase to ensure consistency during analysis.

Tokenization: Tokenization is the process of breaking down the text into individual tokens or words. This step is important for further analysis, as it allows the system to understand the structure of the resume and extract meaningful information. Tokenization can be done using techniques such as splitting on whitespace or using more advanced natural language processing (NLP) algorithms.

Stop Word Removal: Stop words are commonly used words that do not carry much semantic meaning, such as "a," "the," or "and." These words can be removed to reduce noise in the resume data and improve the accuracy of analysis. Stop word removal is

typically done using predefined lists of common stop words or language-specific libraries.

Lemmatization or Stemming: Lemmatization and stemming are techniques used to reduce words to their base or root form. This helps to normalize the text and consolidate variations of the same word. For example, words like "running," "runs," and "ran" can be reduced to their base form "run" through lemmatization or stemming.

CHAPTER-2
LITERATURE SURVEY

The Smart Resume Analyzer is an automated tool designed to analyze resumes and provide feedback to job seekers. It uses natural language processing (NLP) and machine learning techniques to extract information from resumes and evaluate their quality. The literature survey on this topic reveals several key research papers and articles that have contributed to the development and understanding of smart resume analysis. Here are a few notable studies in this field:

"Resume Matching: A Survey" by Guo et al. (2019): This survey paper provides an overview of resume matching techniques and discusses various approaches used to match resumes with job postings. It explores traditional keyword-based matching methods as well as more advanced techniques that utilize NLP and machine learning.

"Automatic Resume Screening: Challenges, Opportunities, and Future Directions" by Ye et al. (2018): This paper presents an in-depth analysis of automatic resume screening systems. It discusses the challenges faced in the field, such as unstructured resume formats and information extraction, and explores potential solutions using NLP and machine learning algorithms.

"Analyzing Job Descriptions and Resumes Using Natural Language Processing" by Lakkaraju et al. (2020): This research paper focuses on the application of NLP techniques for analyzing job descriptions and resumes. It proposes a framework that leverages NLP models to extract relevant skills and experiences from resumes and match them with job requirements.

"Resume Analysis Using Machine Learning" by Ahmed et al. (2020): This paper presents a machine learning-based approach for resume analysis. It discusses the use of feature extraction and classification algorithms to automatically evaluate resumes based on criteria such as qualifications, skills, and experiences.

"Smart Resume Analyzer: A Machine Learning-Based Approach" by Jain and Dhiman (2019): This article introduces the concept of the Smart Resume Analyzer and outlines its architecture. It discusses the use of NLP techniques, feature extraction, and machine learning algorithms for analyzing resumes and providing feedback to job seekers.

CHAPTER-3

NLP

3.1 NLP

NLP stands for Natural Language Processing. It is a field of artificial intelligence (AI) and computational linguistics that focuses on the interaction between computers and human language. NLP aims to enable computers to understand, interpret, and generate human language in a way that is meaningful and useful.

NLP involves the development of algorithms and models that can process and analyze natural language data, such as text or speech. It encompasses a wide range of tasks, including:

Text Understanding: NLP algorithms are used to extract meaning and information from text. This includes tasks such as parsing, part-of-speech tagging, named entity recognition, and sentiment analysis.

Machine Translation: NLP techniques are employed in machine translation systems that can automatically translate text or speech from one language to another.

Information Retrieval: NLP is used to build search engines that can retrieve relevant information from large collections of text documents based on user queries.

Text Generation: NLP models can generate human-like text based on given prompts or conditions. This is used in applications such as chatbots, language generation, and text summarization.

Speech Recognition: NLP techniques are applied in speech recognition systems that convert spoken language into written text. This is used in applications like voice assistants and dictation software.

3.2 WHAT ARE THE GOALS OF NLP

The goals of Natural Language Processing (NLP) are to enable computers to understand, interpret, and generate human language in a way that is meaningful and useful. Some of the key goals of NLP include:

Language Understanding: One of the primary goals of NLP is to develop algorithms and models that can understand and interpret human language. This involves tasks such as parsing sentences, extracting meaning from text, identifying entities and relationships, and comprehending the context of language.

Information Retrieval: NLP aims to develop techniques for effective information retrieval from large collections of textual data. This includes building search engines that can retrieve relevant information based on user queries, and enabling systems to understand user intent and provide accurate and helpful responses.

Machine Translation: NLP focuses on developing systems that can automatically translate text or speech from one language to another. The goal is to build accurate and fluent translation models that bridge language barriers and facilitate communication between people who speak different languages.

Text Generation: NLP aims to create algorithms and models that can generate humanlike text. This includes tasks such as generating coherent and contextually appropriate responses in chatbots, generating summaries of text documents, and creating language models that can produce creative and coherent text.

Sentiment Analysis: NLP techniques are used to analyze and understand the sentiment expressed in text. The goal is to determine the emotions, opinions, and attitudes conveyed in written or spoken language, which can be useful in applications such as social media monitoring, customer feedback analysis, and market research.

These are some of the primary goals of NLP, which aim to bridge the gap between human language and machine understanding. By achieving these goals, NLP enables computers to effectively process, analyze, and generate natural language, opening up a wide range of applications and possibilities in various domains.

3.3 APPLICATIONS OF NLP

Natural Language Processing (NLP) has a wide range of applications across various domains. Some of the key applications of NLP include:

Sentiment Analysis: NLP techniques are used to analyze and understand the sentiment expressed in text, such as social media posts, customer reviews, or survey responses. This information is valuable for businesses to gauge public opinion, conduct market research, and improve customer satisfaction.

Machine Translation: NLP plays a crucial role in machine translation systems, enabling the automatic translation of text or speech from one language to another. This is used in applications such as language localization, cross-language communication, and content translation.

Chatbots and Virtual Assistants: NLP is used to develop conversational agents, such as chatbots and virtual assistants, that can interact with users in a natural language manner. These applications are employed in customer support, information retrieval, and personal assistant services.

3.4 SUBSETS OF NLP

Natural Language Processing (NLP) consists of several subsets or subfields that focus on different aspects of language processing. Some key subsets of NLP include:

Morphological Analysis: This subset deals with the analysis of the internal structure of words, including their inflections, prefixes, suffixes, and stem forms. It involves tasks such as morpheme segmentation, lemmatization, and word normalization.

Syntactic Analysis: Also known as parsing, syntactic analysis focuses on analyzing the grammatical structure of sentences. It involves tasks such as part-of-speech tagging, sentence parsing, and syntactic dependency parsing to understand how words relate to each other in a sentence.

Semantic Analysis: Semantic analysis aims to understand the meaning of words, phrases, and sentences. It involves tasks such as named entity recognition, word sense disambiguation, semantic role labeling, and semantic parsing to capture the semantic relationships between words and extract their meaning.

3.5 TYPES OF NLP:

There are different types or categories of Natural Language Processing (NLP) based on the specific tasks or applications they address. Some common types of NLP include:

Information Retrieval: This type of NLP focuses on retrieving relevant information from large collections of textual data based on user queries. It involves techniques such as keyword matching, relevance ranking, and document retrieval to provide users with relevant documents or passages.

Text Classification: Text classification involves categorizing or labeling text documents into predefined categories or classes. It is used for tasks such as sentiment analysis, topic classification, spam detection, and intent recognition. Machine learning algorithms, such as Naive Bayes, Support Vector Machines (SVM), or deep learning models, are commonly used for text classification.

Named Entity Recognition (NER): NER aims to identify and extract named entities, such as names of people, organizations, locations, or specific dates, from text. It involves tagging or classifying words or phrases into predefined entity categories. NER is used in various applications like information extraction, question answering, and knowledge graph construction.

Sentiment Analysis: Sentiment analysis, also known as opinion mining, focuses on determining the sentiment or opinion expressed in text. It involves classifying text as positive, negative, or neutral, and sometimes detecting more fine-grained emotions. Sentiment analysis is used for social media monitoring, customer feedback analysis, market research, and brand monitoring.

CHAPTER-4
SOFTWARE REQUIREMENTS & ALGORITHMS

4.1 INTRODUCTION TO PYTHON

Python is a widely used high-level programming language, created by Guido Van Rossum and first released in 1991. An interpreted language, Python has a design philosophy that emphasizes code readability (notably using white space indentation to delimit code blocks rather than curly brackets or keywords) and syntax that allows the programmer to express concepts in fewer lines of code that might be used in programming languages such as C++ or Java. It provides constructs that enable to clear programming on both small and large scales.

Python features a dynamic type system & automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Python interpreters are available for many operating systems. The reference implementation of Python is open-source software and has a community-based development model, as do nearly all of its variant implementations. Python is managed by the non-profit Python Software Foundation.

Programming languages have been for ages, and every decade sees the launch of a new language sweeping the developers off their feet. Below are the major features and applications due to which people choose Python as their first programming language:

- Python’s popularity and high salary.
- Python is used in Data Science.
- Python’s scripting and automation.
- Python used with Big Data.
- Python supports Testing.
- Computer Graphics in Python.
- Python used in NLP.
- Python in Web Development.
- Python is portable and extensible.

4.2 HISTORY OF PYTHON

Python was conceived in the late 1980's and its implementation began in December 1989 by Guido van Rossum at Centrum Wickenden and Informatica (CWI) in the Netherlands as a successor to the ABC language (itself inspired by SETL) capable of exception handling and interfacing with the Amoeba operating system.

Guido Van Rossum remains Python's principal author. His continuing central role in Python's development is reflected in the title 'Benevolent Dictator for Life (BDFL)' given to him by the Python community.

Python 2.0 was released on 16 October 2000 and had many major new features, including cycle-detecting garbage collector. With this release, the development process became more transparent.

Python 3.0 (initially called Python 3000 or py3k) was released on 3 December 2008 after a long testing period. It is a major version of the language that is not backward compatible with previous versions. However, many of its major features have been backported to the backward compatible Python 2.6.x and 2.7.x version series.

Python 2.7's end-of-life date (a.k.a. EOL, sunset date) was initially set at 2015, and then postponed to 2020 out of concern that a large body of existing code could not easily be forward ported to Python 3.x.

Python uses duck typing and has typed objects but untyped variable names. Type constraints are not checked at compile time rather, operations on an object may fail, signifying that the given object is not a suitable type. Despite being dynamically typed, Python is strongly typed, forbidding operations that are not well-defined (for example, adding a number to a string) rather than silently attempting to make sense of them.

Python allows the programmer to define their own types using classes, which are most often used for Object-Oriented Programming (OOP). New instances of classes are constructed by calling the class.

4.3 FEATURES OF PYTHON

- **Easy-to-learn:** Python has few keywords, simple structure, and a clearly defined syntax. It allows the student to pick up the language quickly.
- **Easy-to-read:** Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain:** Python's source code is fairly easy-to-maintain.

- **A broad standard library:** Python's bulk library is very portable and crossplatform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode:** Python has support for an interactive mode which allows the interactive testing and debugging of code snippets.
- **Portable:** Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable:** We can add low-level modules to the Python interpreter. These modules enable the programmers to add or customize their tools to be more efficient.
- **Databases:** Python provides interfaces to all major commercial databases.
- **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.
- **Scalable:** Python provides a better structure and support for large programs than shell scripting.
- 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 the large applications.
- It provides very high-level dynamic datatypes and supports dynamic type checking.
- It supports automatic garbage collection.

4.4 DOWNLOADING OF PYTHON

To download Python, we have to follow the below steps:

- Go to www.python.org
- Go to the downloads tab.
- Click on the Python version you required to download according to the OS you are using.
- Double click on the downloaded Python version.
- Select the path for Python.
- Click on install now.
- Click on yes.
- Click on close once it was done.

4.5 INSTALLING OF PYTHON

Python distribution is available for a wide variety of platforms. You need to download only the binary code which is applicable for your platform (OS) and install Python. If the binary code for your platform is not available, you need a C compiler to compile the source code manually. Here is a quick overview of installing Python on Windows platform.

4.6 PYTHON LIBRARIES

Python offers several powerful libraries and frameworks for Natural Language Processing (NLP) tasks. Some popular Python libraries used in NLP are:

NLTK (Natural Language Toolkit): NLTK is a comprehensive library for NLP tasks, providing various functionalities for tokenization, stemming, lemmatization, part-ofspeech tagging, syntactic parsing, and more. It also offers numerous corpora and language resources for experimentation and research.

spaCy: spaCy is a modern and efficient library for NLP in Python. It provides fast and accurate tokenization, part-of-speech tagging, named entity recognition, dependency parsing, and more. spaCy is known for its speed and ease of use, making it suitable for production-level applications.

Gensim: Gensim is a library for topic modeling and document similarity analysis. It provides algorithms such as Latent Semantic Analysis (LSA), Latent Dirichlet Allocation (LDA), and word2vec for word embedding. Gensim is widely used for tasks like topic extraction, document clustering, and word similarity calculation.

scikit-learn: scikit-learn is a general-purpose machine learning library that offers various algorithms and tools for classification, regression, clustering, and dimensionality reduction. It provides support for feature extraction, model evaluation, and pipeline construction, making it useful for NLP tasks like text classification and sentiment analysis.

TensorFlow and Keras: TensorFlow is a popular deep learning framework, and Keras is a highlevel neural network library that runs on top of TensorFlow. Both libraries are widely used for building and training deep learning models, including those for sequence-to-sequence tasks, such as machine translation and text generation.

PyTorch: PyTorch is another popular deep learning framework that provides dynamic computational graphs and extensive support for GPU acceleration. It is widely used in

research and industry for building and training neural networks, including models for natural language processing tasks.

TextBlob: TextBlob is a simplified and user-friendly library built on top of NLTK. It provides a simple API for common NLP tasks, such as tokenization, part-of-speech tagging, noun phrase extraction, sentiment analysis, and more. TextBlob is a great choice for quick prototyping and getting started with NLP.

These are just a few examples of the many Python libraries available for NLP. Each library has its own strengths and focuses on different aspects of NLP tasks. Choosing the right library depends on the specific requirements and goals of your project.

4.7 PROJECT CREATION STEPS

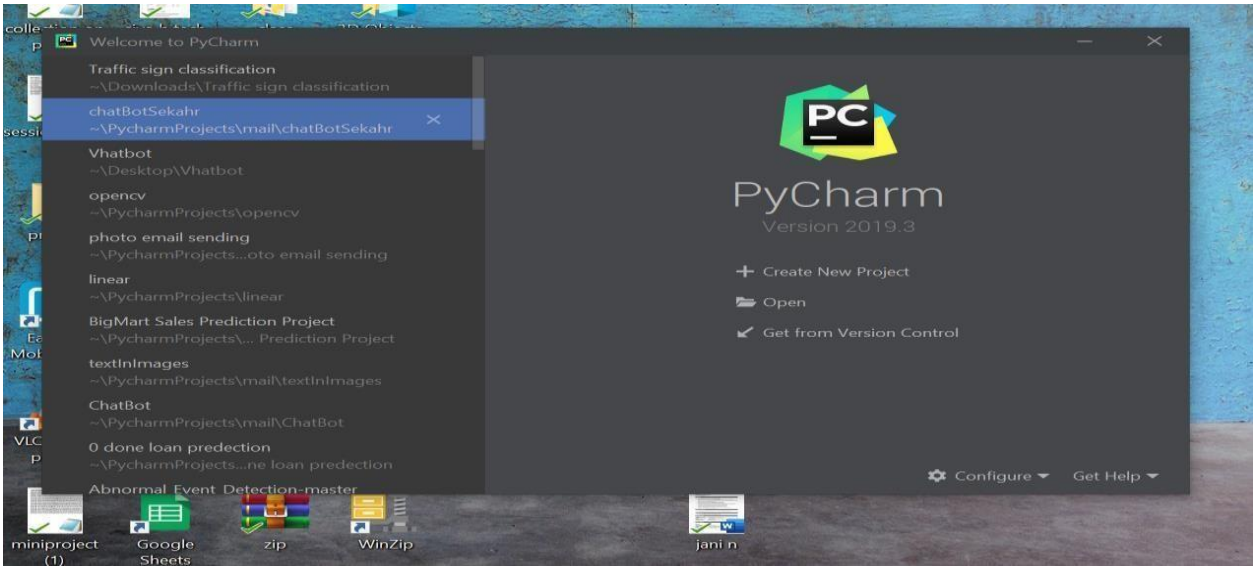


Fig-4.1 Pycharm Tool

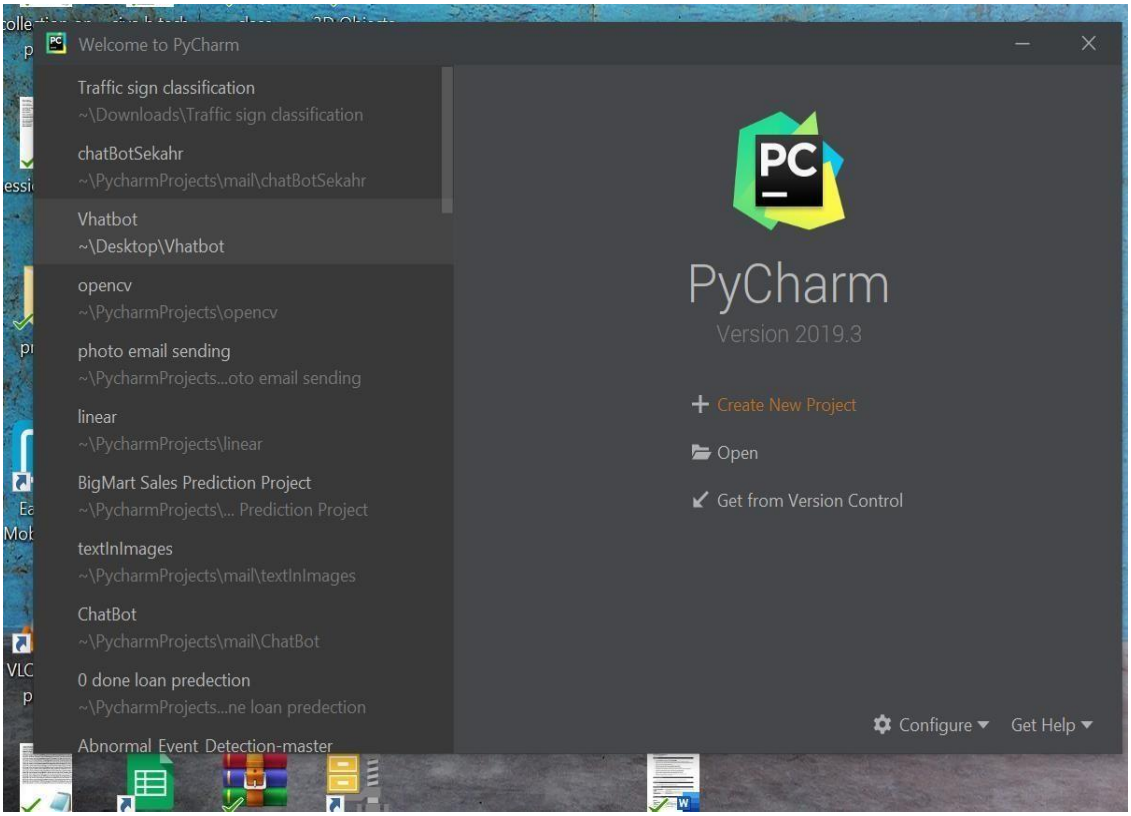


Fig-4.2 Pycharm Projects

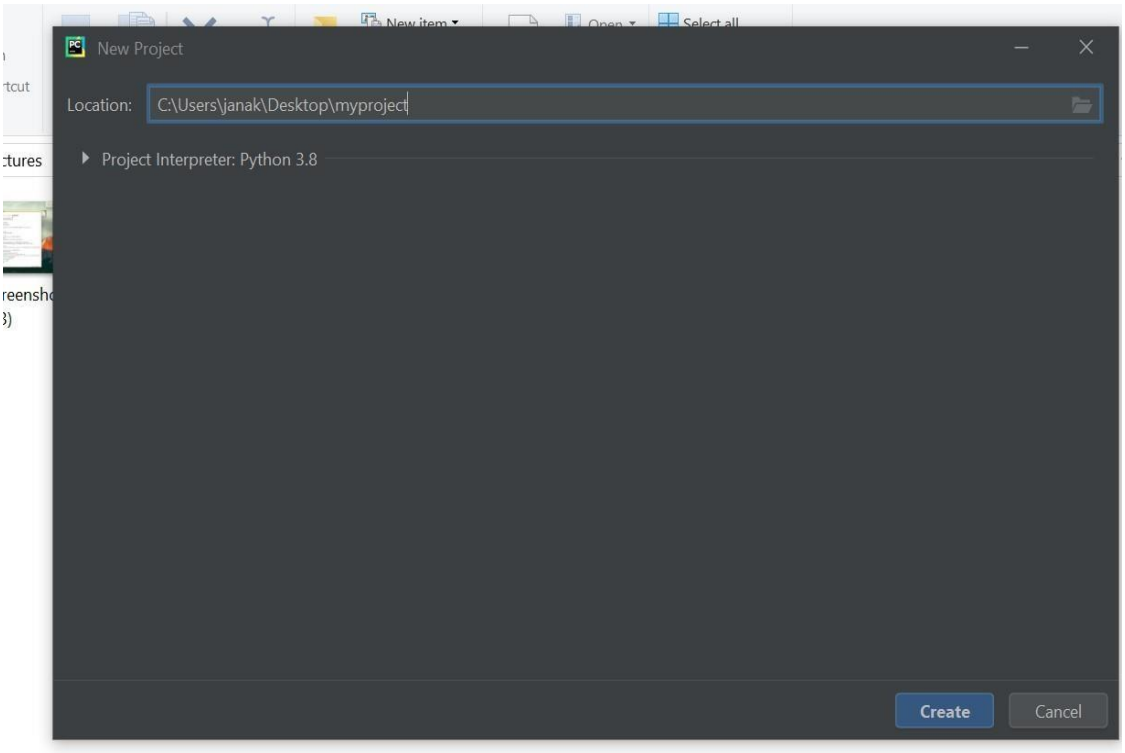


Fig-4.3 Pycharm Files

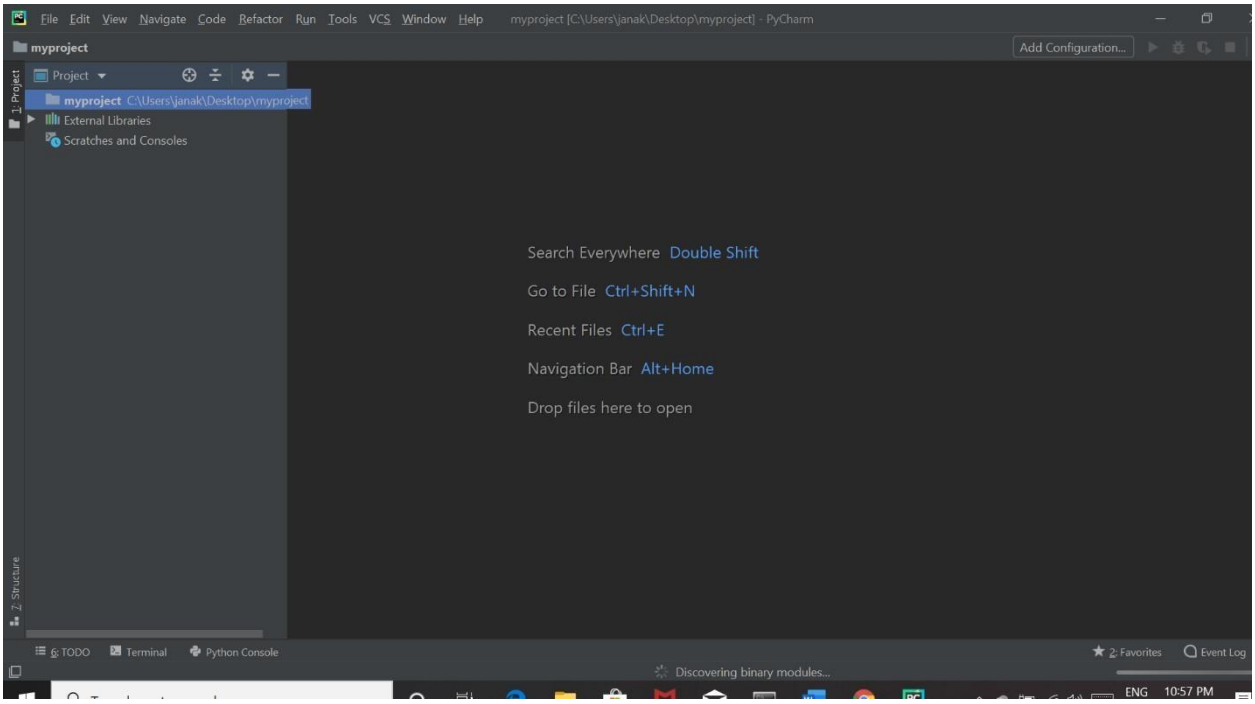


Fig-4.4 Pycharm Myproject

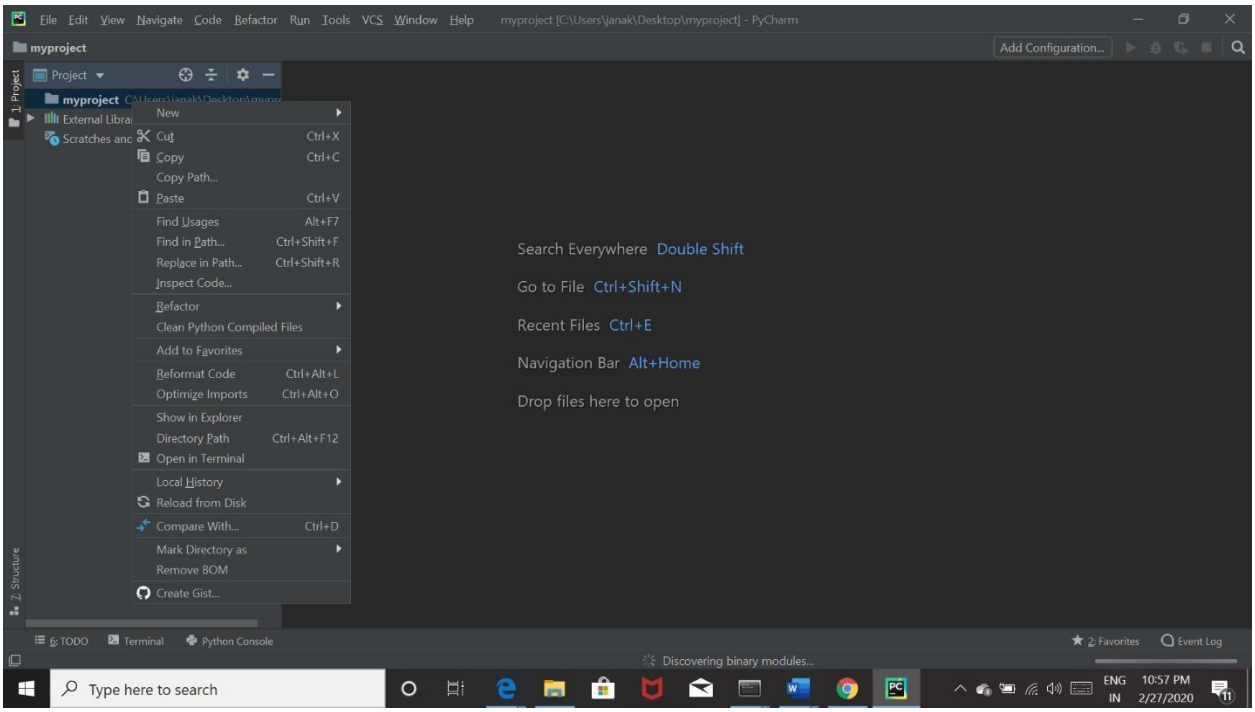


Fig-4.5 PyCharm Tool Activities

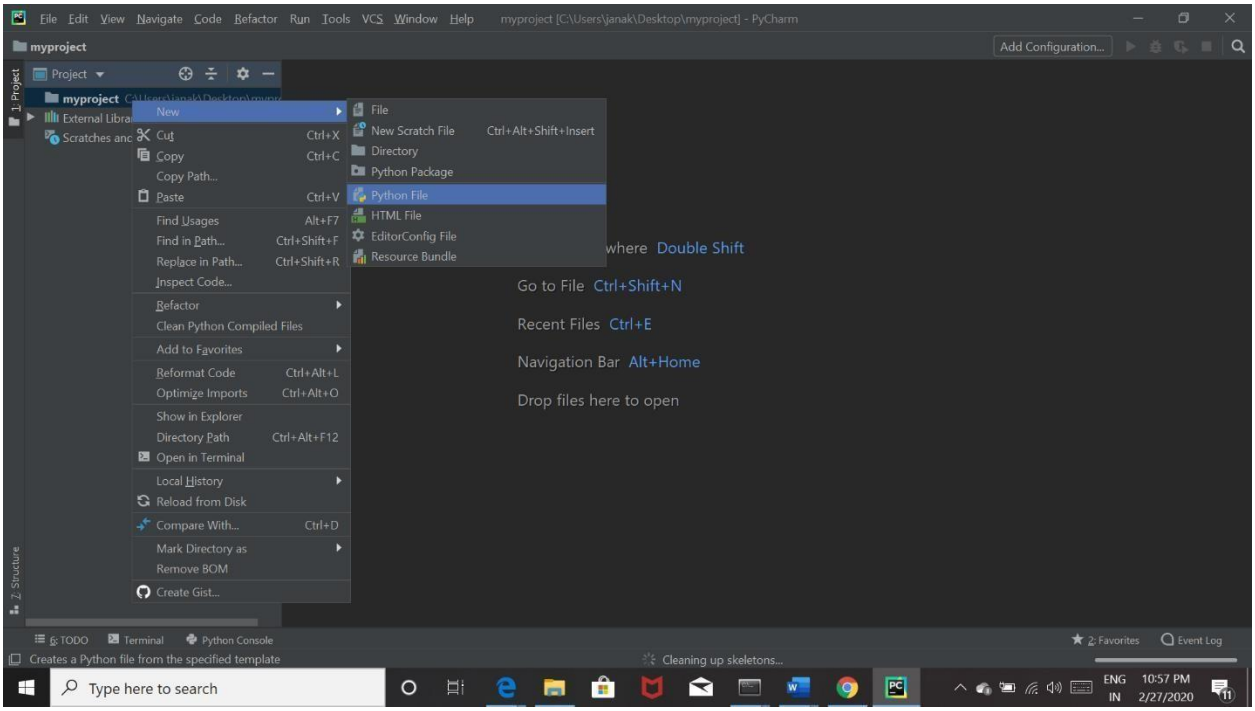


Fig-4.6 Python Files

❼write file name

f

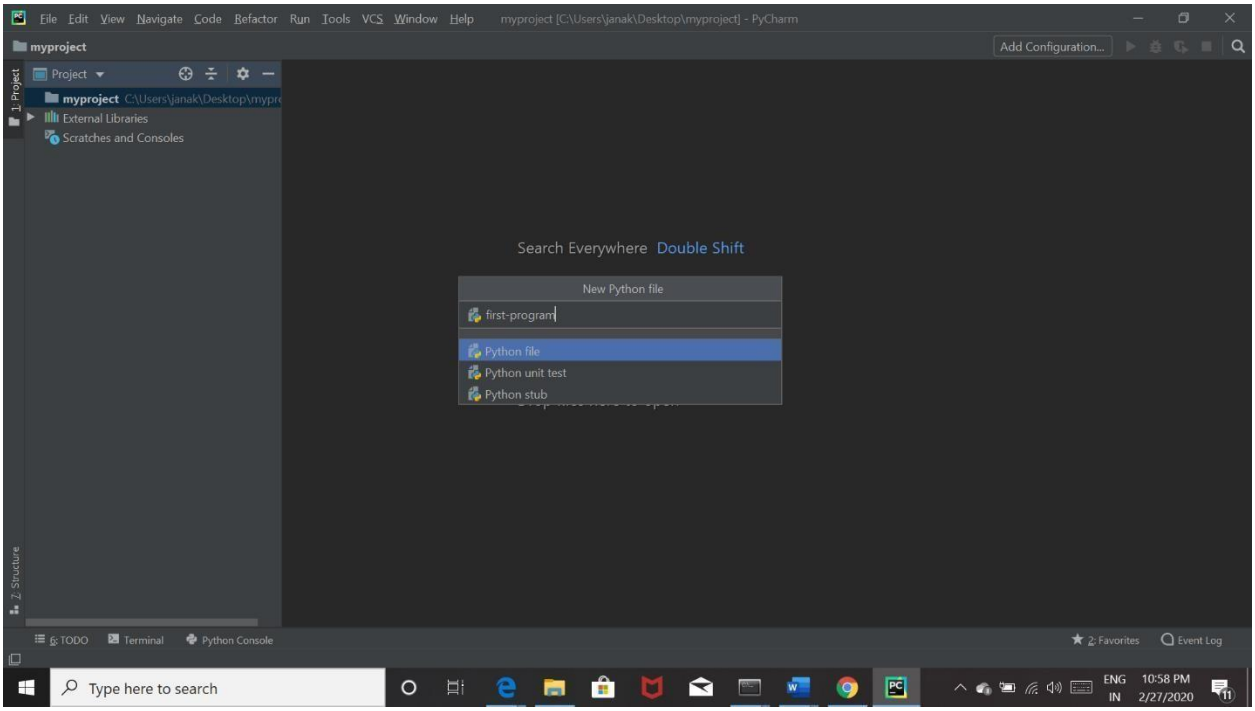


Fig-4.7 Multiple Python Files

CHAPTER-5
SYSTEM DESIGN & TESTING

5.1 INTRODUCTION

The purpose of the design phase is to plan a solution of the problem specified by the requirement document. It is the process of defining software methods, functions, objects and overall structure and interaction of your code so that the resulting functionality will satisfy your users requirements. It allows you to do the best abstraction, to understand the requirements better and meet them better. This prevents redundancy and increases reusability. This phase is the first step in moving from the problem domain to the solution domain. In other words, starting with what is needed design takes us towards how to satisfy the needs. The design of a system is perhaps the most critical factor affecting the quality of the software; it has a major impact on the later phase, particularly testing and maintenance. The output of this phase is the design document. This document is like a blueprint for the solution and is used later during implementation, testing and maintenance. The design activity is often divided into two separate phases System Design and Detailed Design. System Design also called top-level design sign aims to identify the modules that should be in the system, the specifications of these modules, and how they interact with each other to produce the desired results. During detailed design, the internal logic of each of the module's specification in system design is decided.

During this phase, the details of the data are usually specified in a high-level design description language, which is independent of the target language in which the software will eventually be implemented. In system design the focus is on identifying the modules, whereas during detailed design the focus is on designing the logic for each of the modules. During the system design activities, Developers bridge the gap between the requirements specification, produced during requirements elicitation and analysis, and the system that is delivered to the user.

5.2 ALGORITHM & IMPLEMENTATION

Fig:-5.2 Algorithm and Implementation

5.3 ALGORITHM MODEL FLOW DIAGRAM

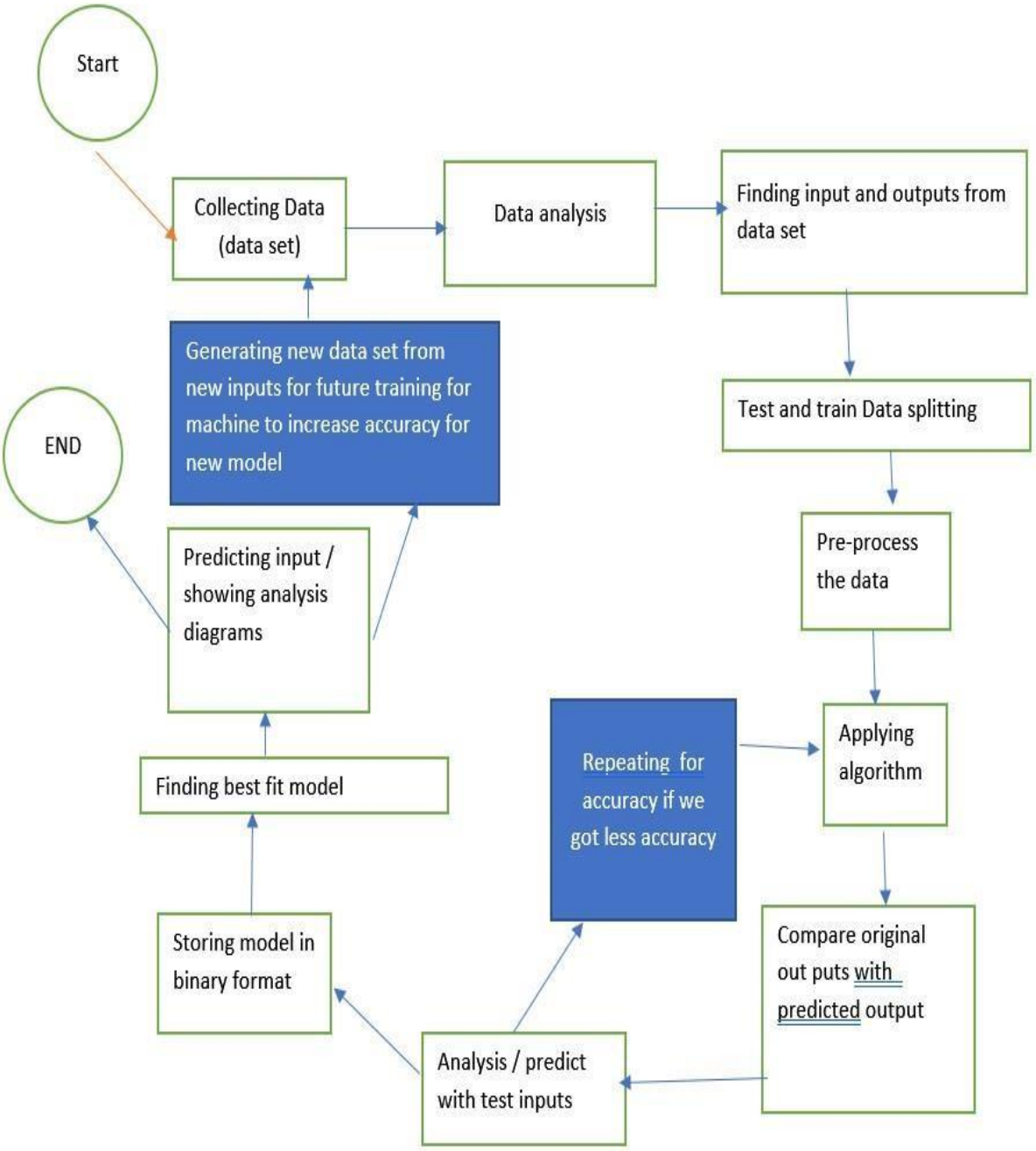


Fig:-5.3 Algorithm Model Flow Diagram

5.4 Sample code: Main.py

```
import streamlit as st
import pandas as pd
import base64,random
import time,datetime from pyresparser
import ResumeParser from pdfminer3.layout
import LAParams, LTTextBox from pdfminer3.pdfpage
import PDFPage from pdfminer3.pdfinterp
 import PDFResourceManager from pdfminer3.pdfinterp
import
PDFPageInterpreter from pdfminer3.converter import TextConverter import
io,random from streamlit_tags import st_tags from PIL import Image import pymysql
from Courses import
ds_course,web_course,android_course,ios_course,uiux_course,resume_videos,intervi
ew_
vide os import pafy import plotly.express as px  import youtube_dl

def     fetch_yt_video(link):
video = pafy.new(link)
    return video.title

st.set_page_config(
page_title="Smart Resume Analyzer",
page_icon='./Logo/SRA_Logo.ico',    layout="wide"
)

def get_table_download_link(df,filename,text):
    csv = df.to_csv(index=False)
    b64 = base64.b64encode(csv.encode()).decode()    # some strings <-> bytes
conversions
necessary here
    # href = f'<a href="data:file/csv;base64,{b64}">Download Report</a>'    href = f'<a
href="data:file/csv;base64,{b64}" download="{filename}">{text}</a>'
```

```
    return href

def pdf_reader(file):
    resource_manager = PDFResourceManager()
    fake_file_handle = io.StringIO()     converter = TextConverter(resource_manager,
fake_file_handle,      laparams=LAParams())                    page_interpreter      =
PDFPageInterpreter(resource_manager, converter)
    with open(file, 'rb') as fh:
        for page in PDFPage.get_pages(fh,
                                      caching=True,
check_extractable=True):            page_interpreter.process_page(page)
            print(page)
        text = fake_file_handle.getvalue()

    #close open handles:-     converter.close()     fake_file_handle.close()     return text

def       show_pdf(file_path):     with open(file_path, "rb") as
f:
        base64_pdf  =  base64.b64encode(f.read()).decode('utf-8')            pdf_display  =
F'<iframe src="data:application/pdf;base64,{base64_pdf}"
width="700"                height="1000"                type="application/pdf"></iframe>'
st.markdown(pdf_display, unsafe_allow_html=True)

def course_recommender(course_list):
    st.subheader("**Courses & Certificates  Recommendations**")      c = 0
rec_course = []     no_of_reco = st.slider('Choose Number of Course
Recommendations:', 1, 10, 4)     random.shuffle(course_list)     for c_name, c_link in
course_list:
        c += 1          st.markdown(f"({c}) [{c_name}]({c_link})")
rec_course.append(c_name)         if c == no_of_reco:
            break      return rec_course

connection                    =                    pymysql.connect(host='localhost',port=3307,
user='root',password='',db='sra') cursor = connection.cursor()
```

```
def
insert_data(name,email,res_score,timestamp,no_of_pages,reco_field,cand_level,skills
,re com mended_skills,courses):    DB_table_name = 'user_data'    insert_sql = "insert
into
" + DB_table_name + """    values (0,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)"""
   rec_values = (name, email, str(res_score), timestamp,str(no_of_pages),
reco_field, cand_level, skills,recommended_skills,courses)   cursor.execute(insert_sql,
rec_values)    connection.commit()


def run():
   st.title("Smart Resume Analyser ")    st.sidebar.markdown("# Choose User")
activities = ["Normal User", "Admin"]    choice = st.sidebar.selectbox("Choose
among the given options:", activities)    img = Image.open('SRA_Logo.jpg')    img =
img.resize((250,250))    st.image(img)

 # Create the DB:-
   db_sql = """CREATE DATABASE IF NOT EXISTS SRA;"""
   cursor.execute(db_sql)

# Create table:-
   DB_table_name = 'user_data'    table_sql = "CREATE TABLE IF NOT
EXISTS " + DB_table_name + """
                (ID INT NOT NULL AUTO_INCREMENT,
                 Name varchar(100) NOT NULL,                Email_ID VARCHAR(50)
NOT NULL,                resume_score VARCHAR(8) NOT NULL,
Timestamp VARCHAR(50) NOT NULL,
                 Page_no VARCHAR(5) NOT NULL,
                 Predicted_Field VARCHAR(25) NOT NULL,
                 User_level VARCHAR(30) NOT NULL,
                 Actual_skills          VARCHAR(300)          NOT          NULL,
Recommended_skills VARCHAR(300) NOT NULL,
                 Recommended_courses VARCHAR(600) NOT NULL,
```

```
                PRIMARY KEY (ID));
                """
    cursor.execute(table_sql)

    if choice == 'Normal User':
        pdf_file = st.file_uploader("Choose your Resume", type=["pdf"])          if pdf_file
is not None:              with
st.spinner('Uploading your Resume....'):
            time.sleep(4)
            save_image_path  =  './Uploaded_Resumes/'+pdf_file.name                    with
open(save_image_path, "wb") as f:
                f.write(pdf_file.getbuffer())
            show_pdf(save_image_path)            resume_data =
ResumeParser(save_image_path).get_extracted_data()                    #  parser  =
ResumeParser('path/to/resume.pdf', config_path='path/to/config.cfg')
            if resume_data:
                ##  Get  the  whole  resume  data  resume_text =
pdf_reader(save_image_path)

                st.header("**Resume Analysis**")
st.success("Hello "+ resume_data['name'])                st.subheader("**Your Basic
info**")
                try:
                    st.text('Name: '+resume_data['name'])
st.text('Email: ' + resume_data['email'])
st.text('Contact: ' + resume_data['mobile_number'])                st.text('Resume pages:
'+str(resume_data['no_of_pages']))
                except:
pass
cand_level = ''
                if resume_data['no_of_pages'] == 1:
                    cand_level = "Fresher"
```

```
            st.markdown( '''<h4 style='text-align: left; color: #d73b5c;'>You are looking Fresher.</h4>''',unsafe_allow_html=True) elif resume_data['no_of_pages'] == 2:
                cand_level = "Intermediate"
                st.markdown('''<h4 style='text-align: left; color: #1ed760;'>You are at intermediate level!</h4>''',unsafe_allow_html=True)
            elif resume_data['no_of_pages'] >=3:
                cand_level = "Experienced"
                st.markdown('''<h4 style='text-align: left; color: #fba171;'>You are at experience level!''',unsafe_allow_html=True)

            st.subheader("**Skills Recommendation **") #Skills from Resume:-
keywords = st_tags(label='### Skills that you have', text='See our skills recommendation', value=resume_data['skills'],key = '1')

            #Recommendation Skills:-
            ds_keyword = ['tensorflow','keras','pytorch','machine learning','deep Learning','flask','streamlit']
            web_keyword = ['react', 'django', 'node jS', 'react js', 'php', 'laravel', 'magento', 'wordpress',
                           'javascript', 'angular js', 'c#', 'flask']
            android_keyword = ['android','android development','flutter','kotlin','xml','kivy']
ios_keyword = ['ios','ios development','swift','cocoa','cocoa touch','xcode']
            uiux_keyword = ['ux','adobe xd','figma','zeplin','balsamiq','ui','prototyping','wireframes','storyframes','adobe photoshop','photoshop','editing','adobe illustrator','illustrator','adobe after effects','after effects','adobe premier pro','premier pro','adobe indesign','indesign','wireframe','solid','grasp','user research','user experience']


            recommended_skills = []
            reco_field = ''
rec_course = ''             ##
```

```
Courses recommendation:-
            for i in resume_data['skills']:                ## Data science recommendation:-
if i.lower() in ds_keyword:                print(i.lower())                reco_field =
'Data Science'                st.success("** Our analysis says you are looking for
Data Science Jobs.**")
                recommended_skills = ['Data Visualization','Predictive
Analysis','Statistical
Modeling','Data Mining','Clustering & Classification','Data Analytics','Quantitative
Analysis','Web Scraping','ML
Algorithms','Keras','Pytorch','Probability','Scikitlearn','Tensorflow',"Flask",'Streamlit']
recommended_keywords = st_tags(label='### Recommended skills for you.',
                text='Recommended skills generated from
System',value=recommended_skills,key = '2')
                st.markdown('''<h4 style='text-align: left; color: #1ed760;'>Adding this
skills to resume will boost the chances of getting a Job
</h4>''',unsafe_allow_html=True)
                rec_course = course_recommender(ds_course)                break

            ## Web development recommendation:-            elif i.lower() in
web_keyword:
                print(i.lower())
                reco_field = 'Web Development'
                st.success("** Our analysis says you are looking for Web Development
Jobs
**")
                recommended_skills = ['React','Django','Node JS','React
JS','php','laravel','Magento','wordpress','Javascript','Angular JS','c#','Flask','SDK']
recommended_keywords = st_tags(label='### Recommended skills for you.',
                text='Recommended skills generated from
System',value=recommended_skills,key = '3')
                st.markdown('''<h4 style='text-align: left; color: #1ed760;'>Adding this
skills to resume will boost the chances of getting a Job
</h4>''',unsafe_allow_html=True)
                rec_course = course_recommender(web_course)
```

```
                break

            ## Android App Development:-          elif i.lower() in android_keyword:
                print(i.lower())
                reco_field = 'Android Development'
                st.success("** Our analysis says you are looking for Android App Development Jobs **")          recommended_skills = ['Android','Android development','Flutter','Kotlin','XML','Java','Kivy','GIT','SDK','SQLite']
recommended_keywords = st_tags(label='### Recommended skills for you.',
                text='Recommended skills generated from System',value=recommended_skills,key = '4')
                st.markdown("'<h4 style='text-align: left; color: #1ed760;'>Adding this skills to resume will boost the chances of getting a Job </h4>'",unsafe_allow_html=True)
                rec_course = course_recommender(android_course)
                break

            ## IOS App Development:-          elif i.lower() in ios_keyword:
                print(i.lower())
                reco_field = 'IOS Development'
                st.success("** Our analysis says you are looking for IOS App Development Jobs **")
                recommended_skills = ['IOS','IOS Development','Swift','Cocoa','Cocoa Touch','Xcode','Objective-C','SQLite','Plist','StoreKit',"UI-Kit",'AV Foundation','AutoLayout']          recommended_keywords = st_tags(label='### Recommended skills for you.',
                text='Recommended skills generated from System',value=recommended_skills,key = '5')
                st.markdown("'<h4 style='text-align: left; color: #1ed760;'>Adding this skills to resume will boost the chances of getting a Job </h4>'",unsafe_allow_html=True)
                rec_course = course_recommender(ios_course)
                break
```

```
## Ui-UX Recommendation:-        elif i.lower() in uiux_keyword:
    print(i.lower())
    reco_field = 'UI-UX Development'
    st.success("** Our analysis says you are looking for UI-UX
Development Jobs  **")
    recommended_skills = ['UI','User Experience','Adobe
XD','Figma','Zeplin','Balsamiq','Prototyping','Wireframes','Storyframes','Adobe
Photoshop','Editing','Illustrator','After Effects','Premier
Pro','Indesign','Wireframe','Solid','Grasp','User                Research']
recommended_keywords = st_tags(label='### Recommended skills for you.',
    text='Recommended    skills    generated    from
System',value=recommended_skills,key = '6')
    st.markdown("'<h4 style='text-align: left; color: #1ed760;'>Adding this
skills  to  resume  will  boost  the  chances  of  getting  a  Job
</h4>'",unsafe_allow_html=True)
    rec_course = course_recommender(uiux_course)
    break

# Insert into table:-        ts = time.time()        cur_date =
datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d')        cur_time =
datetime.datetime.fromtimestamp(ts).strftime('%H:%M:%S')        timestamp =
str(cur_date+'_'+cur_time)

#Resume  writing  recommendation:-
st.subheader("**Resume Tips & Ideas **")
resume_score  =  0
if 'Objective' in resume_text:
    resume_score = resume_score+20
    st.markdown("'<h4 style='text-align: left; color: #1ed760;'>[+] Awesome!
You have added Objective</h4>'",unsafe_allow_html=True)
else:
    st.markdown("'<h4 style='text-align: left; color: #fabc10;'>[-] According
to
```

```
our recommendation please add your career objective, it will give your career intension
to the Recruiters.</h4>'",unsafe_allow_html=True)

        if 'Declaration'  in resume_text:
            resume_score = resume_score + 20
            st.markdown("'<h4 style='text-align: left; color: #1ed760;'>[+] Awesome!
You have added Delcaration✍/h4>'",unsafe_allow_html=True)            else:
            st.markdown("'<h4 style='text-align: left; color: #fabc10;'>[-] According
to
our recommendation please add Declaration✍. It will give the assurance that
everything written on your resume is true and fully acknowledged by
you</h4>'",unsafe_allow_html=True)

        if  'Hobbies'  or  'Interests'in  resume_text:
resume_score = resume_score + 20
            st.markdown("'<h4 style='text-align: left; color: #1ed760;'>[+] Awesome!
You have added your Hobbies </h4>'",unsafe_allow_html=True)            else:
            st.markdown("'<h4 style='text-align: left; color: #fabc10;'>[-] According
to
our recommendation please add Hobbies . It will show your persnality to the
Recruiters and give the assurance that you are fit for this role or
not.</h4>'",unsafe_allow_html=True)

        if 'Achievements' in resume_text:
            resume_score = resume_score + 20
            st.markdown("'<h4 style='text-align: left; color: #1ed760;'>[+] Awesome!
You have added your Achievements  </h4>'",unsafe_allow_html=True)            else:
            st.markdown("'<h4 style='text-align: left; color: #fabc10;'>[-] According
to
our recommendation please add Achievements . It will show that you are capable for
the required position.</h4>'",unsafe_allow_html=True)

        if 'Projects' in resume_text:
```

```
            resume_score = resume_score + 20
            st.markdown("'<h4 style='text-align: left; color: #1ed760;'>[+] Awesome!
You have added your Projects  </h4>'",unsafe_allow_html=True)            else:
            st.markdown("'<h4 style='text-align: left; color: #fabc10;'>[-] According
to
our recommendation please add Projects . It will show that you have done work
related the required position or not.</h4>'",unsafe_allow_html=True)
st.subheader("**Resume Score **")            st.markdown(
            """
            <style>
               .stProgress > div > div > div > div {                    background-color:
#d73b5c;
               }
            </style>""",
            unsafe_allow_html=True,
         )
      my_bar = st.progress(0)            score = 0            for percent_complete in
range(resume_score):
            score                +=1
time.sleep(0.1)
            my_bar.progress(percent_complete          +          1)
st.success('** Your Resume Writing Score: ' + str(score)+'**')
      st.warning("** Note: This score is calculated based on the content that you have
added in your Resume. **")
      st.balloons()

      insert_data(resume_data['name'],      resume_data['email'],  str(resume_score),
timestamp,
                  str(resume_data['no_of_pages']), reco_field,
cand_level, str(resume_data['skills']),                  str(recommended_skills),
str(rec_course))
      ## Resume writing video:-          st.header("**Bonus
Video for Resume Writing Tips **")          resume_vid =
```

```
random.choice(resume_videos)  res_vid_title = fetch_yt_video(resume_vid)
st.subheader("
**"+res_vid_title+"**")  st.video(resume_vid)

    ## Interview Preparation Video:-  st.header("**Bonus Video for
Interview  Tips **")  interview_vid = random.choice(interview_videos)
int_vid_title = fetch_yt_video(interview_vid)  st.subheader("  **" +
int_vid_title + "**")  st.video(interview_vid)

    connection.commit()  else:  st.error('Something went wrong..')
  else:
    # Admin Side:-  st.success('Welcome to Admin Side')  ad_user =
st.text_input("Username")  ad_password = st.text_input("Password",
type='password')
    if st.button('Login'):  if ad_user == 'SRA_NLP'
and ad_password == 'nlp123':  st.success("Welcome Admin")
        # Display Data:-
        cursor.execute("'SELECT*FROM user_data"')  data  =
    cursor.fetchall()
st.header("**User's  Data**")
        df = pd.DataFrame(data, columns=['ID', 'Name', 'Email', 'Resume Score',
'Timestamp', 'Total Page',
                                         'Predicted  Field', 'User  Level', 'Actual
    Skills', 'Recommended Skills',
                                         'Recommended Course'])  st.dataframe(df)
        st.markdown(get_table_download_link(df,'User_Data.csv','Download
Report'),
unsafe_allow_html=True)
         ## Admin Side Data:-  query = 'select * from user_data;'
plot_data = pd.read_sql(query, connection)

        ## Pie chart for predicted field recommendations:-  labels =
plot_data.Predicted_Field.unique()
        print(labels)
```

```
        values = plot_data.Predicted_Field.value_counts()  print(values)
st.subheader("  **Pie-Chart for Predicted Field Recommendations**")  fig
= px.pie(df, values=values, names=labels, title='Predicted Field according to
the Skills')  st.plotly_chart(fig)

        ### Pie chart for User's  Experienced Level:-  labels =
plot_data.User_level.unique()  values =
plot_data.User_level.value_counts()  st.subheader("  ** PieChart for User's
Experienced Level**")
        fig = px.pie(df, values=values, names=labels, title="Pie-Chart  for User's
Experienced Level")
        st.plotly_chart(fig)

else:
        st.error("Wrong ID & Password Provided")
run()
```

5.5 NLP Testing

NLP testing refers to the process of evaluating and validating the performance and quality of Natural Language Processing (NLP) models, algorithms, or systems. Testing is essential to ensure that NLP applications function accurately, produce reliable results, and meet the desired objectives. Here are some key considerations for NLP testing:

Test Data: Gather or create test datasets that cover a diverse range of language patterns, variations, and scenarios relevant to your NLP application. The test data should represent realworld use cases and cover both expected and edge cases.

Test Coverage: Ensure that your test suite covers different aspects of NLP, including tokenization, part-of-speech tagging, named entity recognition, syntactic parsing, semantic analysis, sentiment analysis, and any other specific tasks your NLP system performs.

Performance Testing: Evaluate the performance of your NLP system in terms of speed, memory usage, and scalability. Test the system with varying sizes of input data to ensure it performs efficiently under different workloads.

Accuracy and Precision: Measure the accuracy and precision of your NLP models or algorithms by comparing their outputs with manually annotated or gold standard data. Use evaluation metrics such as precision, recall, F1-score, or accuracy to assess the quality of results.

Error Analysis: Analyze and understand the types of errors made by your NLP system. Identify common failure cases, limitations, and areas where the system needs improvement. Error analysis helps in refining the models, addressing biases, and making targeted enhancements.

Robustness Testing: Test the robustness of your NLP system by introducing variations, noise, misspellings, or grammatical errors in the input text. This helps assess the system's ability to handle noisy or imperfect data and still provide meaningful results.

Language Coverage: If your NLP system supports multiple languages, ensure that testing is done on diverse language datasets to validate the language-specific performance and handle language-specific challenges effectively.

Integration Testing: If your NLP system is part of a larger application or pipeline, perform integration testing to ensure seamless interaction with other components and validate the overall system behavior.

User Acceptance Testing: Involve end-users or domain experts in the testing process to gather feedback on the usability, effectiveness, and satisfaction of the NLP application.

Incorporate user feedback to improve the system's performance and user experience.

Continuous Testing: Implement continuous testing practices to automate the testing process and ensure that your NLP system remains robust and reliable even as new features or updates are introduced.

NLP testing is an iterative process that involves refining and enhancing the models or algorithms based on the insights gained from testing. It helps identify and address limitations.

CHAPTER-6

OUTPUT SCREENSHOTS

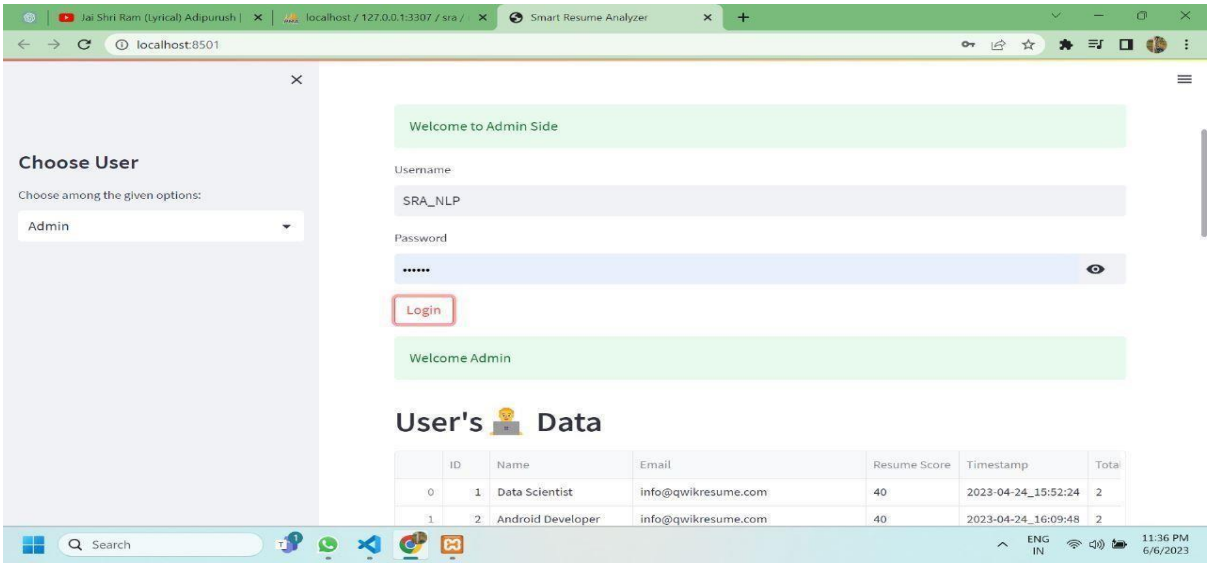


Fig.6.1 Project Interface

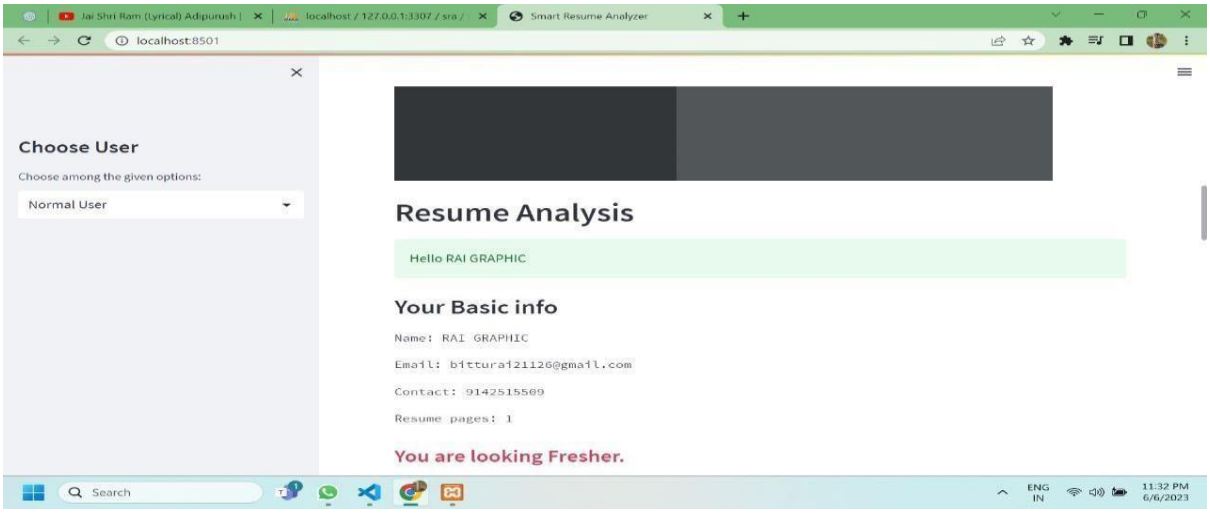


Fig 6.2 Resume Data

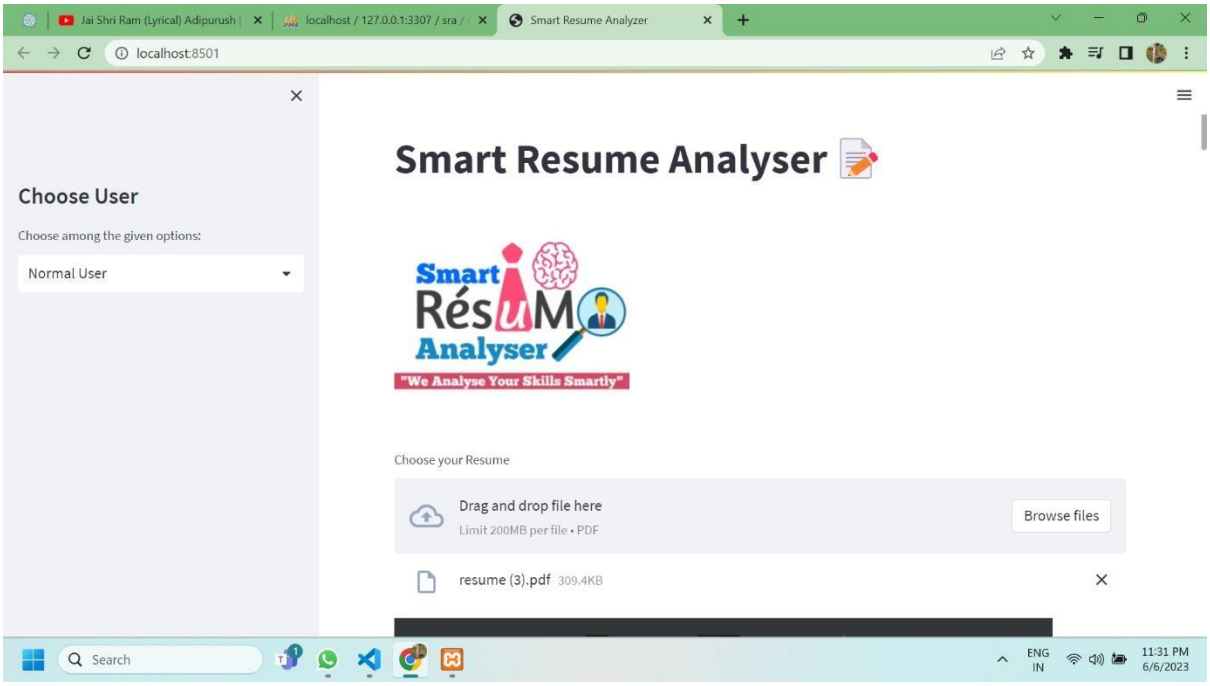


Fig-6.3 Resume Uploading

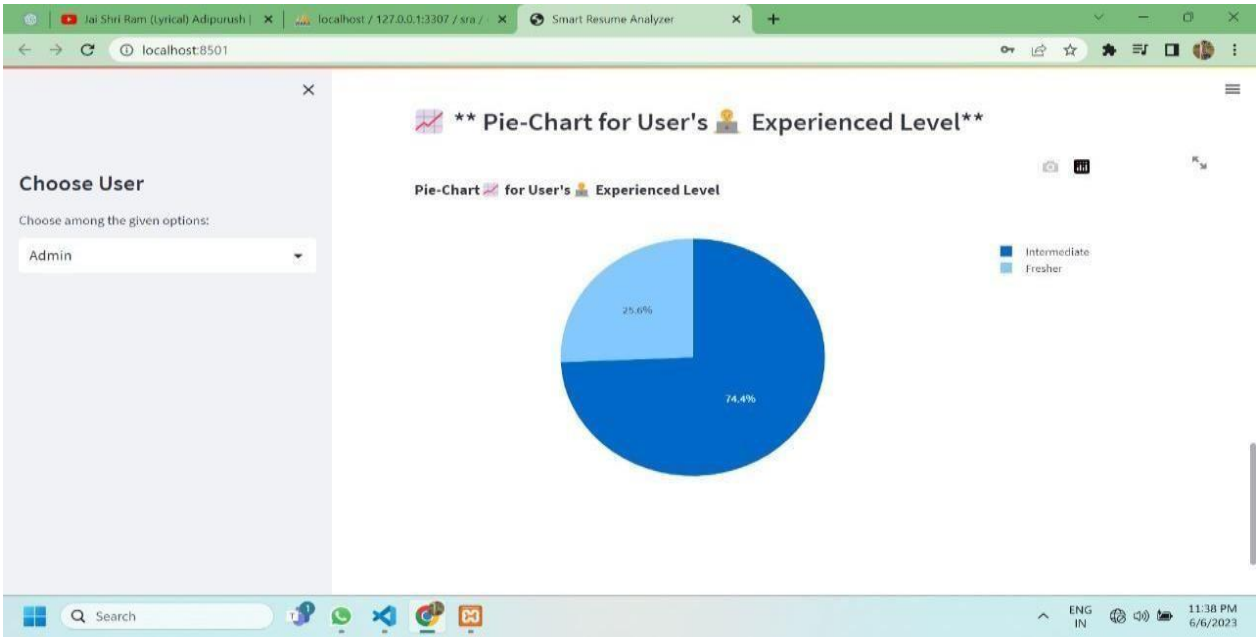


Fig-6.4 Users Resume

CHAPTER-7
CONCULSUION

In conclusion, the Smart Resume Analyzer is an innovative application that leverages Natural Language Processing (NLP) techniques to analyze and evaluate resumes effectively. By utilizing various NLP tasks such as named entity recognition, syntactic parsing, and sentiment analysis, the analyzer can extract key information, assess the quality of the resume, and provide valuable insights to both job seekers and recruiters.

The Smart Resume Analyzer offers several benefits. For job seekers, it helps identify areas of improvement, highlights strengths and weaknesses, and provides recommendations to enhance the resume's content and structure. It assists in tailoring resumes to specific job requirements, increasing the chances of getting noticed by potential employers.

For recruiters and hiring managers, the analyzer saves time and effort by automating the initial screening process. It efficiently filters through a large number of resumes, identifies qualified candidates, and provides a comprehensive analysis of their qualifications, skills, and experience. This streamlines the recruitment process and improves the efficiency of talent acquisition.

The analyzer's use of NLP techniques ensures accurate and detailed analysis of resumes. It can identify relevant entities such as candidate names, educational institutions, work experience, and skills, providing a comprehensive overview of the candidate's profile. Additionally, sentiment analysis can gauge the overall tone and sentiment expressed in the resume, offering insights into the candidate's attitude and professionalism.

However, it is important to acknowledge that the Smart Resume Analyzer has its limitations. While NLP algorithms have improved significantly, they are not perfect and can occasionally make errors or misinterpretations. The accuracy of the analysis depends on the quality of the NLP models and the training data used. It is crucial to periodically update and fine-tune the models to ensure optimal performance and adaptability to changing resume formats and language patterns.

Overall, the Smart Resume Analyzer demonstrates the power of NLP in transforming the recruitment process. By automating resume analysis and providing valuable insights, it streamlines the hiring process for both job seekers and recruiters. further advancements in NLP technology, the analyzer has the potential to revolutionize.

CHAPTER-8

BIBLIOGRAPHY

[1]"Smart Resume Parsing Using Natural Language Processing Techniques" by M. Siva Sankar and Dr. M. Hemalatha:
URL:https://www.researchgate.net/publication/316899151_Smart_Resume_Parsing_Using_N atural_ Language_Processing_Techniques

[2]"Automated Resume Screening and Recommendation System" by Anjali Kulkarni and Aniruddha Shitole:
Link: https://ieeexplore.ieee.org/document/8798884

[3]"Automated Resume Classification and Shortlisting using Machine Learning" by Prithvi raj Bhosale and Dr. Rahul J. Karalee.
URL:https://www.researchgate.net/publication/336243305_Automated_Resume_Classificatio n_and_ Shortlisting_using_Machine_Learning)

[4]"Resume Ranker: A Framework for Smart Resume Ranking and Shortlisting" by Satish Kumar and Dr. Neetu Sardana.
Link: https://link.springer.com/chapter/10.1007/978-981-15-0991-3_23

[5]"Deep Learning-Based Resume Screening" by Shao-Wen Yang and Chin-Laing Lei.
Link: https://www.researchgate.net/publication/331647684_Deep_Learning-Based_Resume_Screening

[6]"Automated Resume Screening using Machine Learning Techniques" by Himanshu Singhal and Dr. Sanjeev Jain.
Link: https://ieeexplore.ieee.org/document/8479300

[7]"Efficient Filtering of Job Applicants Using Automated Resume Analysis" by Nazia Majeed, Dr. R. R. Hanchinal, and Dr. T. V. Suresh Kumar.
Link: https://ieeexplore.ieee.org/document/8628670