## ABSTRACT

This study gives an efficient and interactive web application for real time dynamic resume screening with flexible job description. Our model takes the job description and resumes as input and classifies the best match for the resume based on the given job description. This is constructed using Natural language processing ( text pre processing )and ( Topic Modeling ) Latent Dirichlet Allocation to get the best match of all resumes. We have used different similarity indexes and did average so that we get an accurate bound of score and rank it accurately. Since many big tech companies are facing issues due to the huge volume of data ( BIG Data ) and difficult to manage, keeping this in mind we created a streamlit interactive web application dashboard that is used to solve the resume screening with different job descriptions irrespective of the huge amount of data.

**Keywords** : NLP , Resume , Topic Modeling , LDA , Machine Learning , Text pre processing

# Chapter - 01

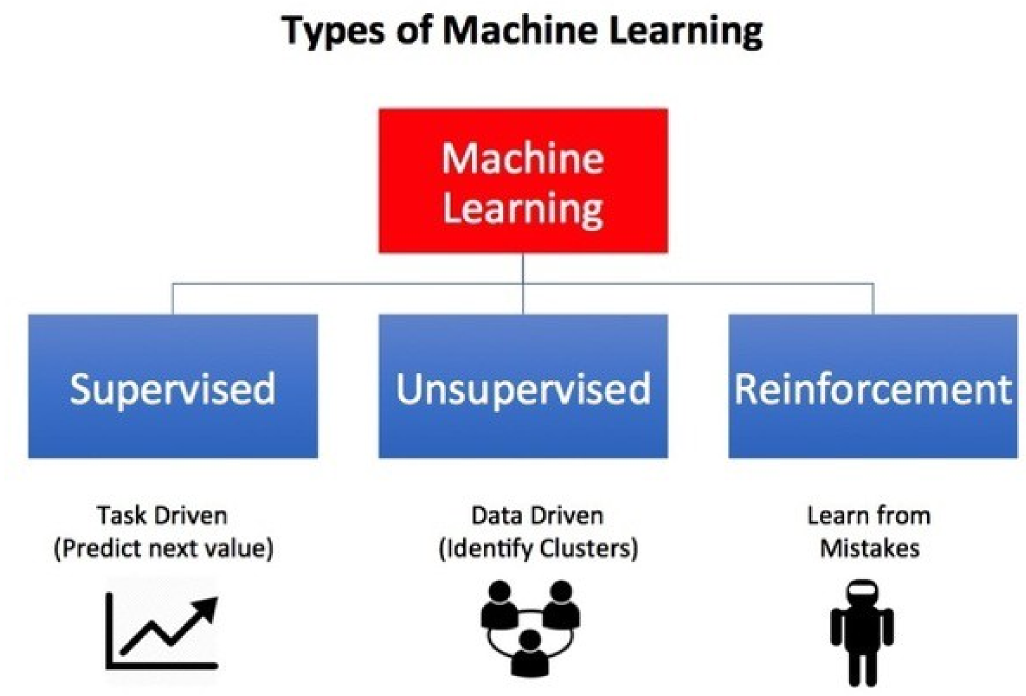
# INTRODUCTION

## 1.1 overview of the project

Choosing the right people for the job is the biggest responsibility of every business since choosing the right set of people can accelerate business growth exponentially.We will discuss here an example of such a business, which we know as the IT department. We know that the IT department falls short of growing markets.Due to many big projects with big companies, their team does not have time to read resumes and choose the best resume according to their requirements.To solve this type of problem, the company always chooses a third party whose job is to make the resume as per the requirement. These companies are known by the name of Hiring Service Organization. It’s all about the information resume screen.The work of selecting the best talent, assignments, online coding contests among many others is also known as resume screen.Due to lack of time, big companies do not have enough time to open resumes, due to which they have to take the help of any other company. For which they have to pay money. Which is a very serious problem.To solve this problem, the company wants to start the work of the resume screen itself so keeping this in mind we have build an interactive web application dashboard that is used to classify the resumes based on the job descriptions.

## 1.2 INTRODUCTION TO MACHINE LEARNING

Machine Learning is used anywhere from automating mundane tasks to offering intelligent insights, industries in every sector try to benefit from it. You may already be using a device that utilizes it. For example, a wearable fitness tracker like Fitbit, or an intelligent home assistant like Google Home. But there are much more examples of ML in useMachine Learning algorithms enable the computers to learn from data, and even improve themselves, without being explicitly programmed.Machine learning (ML) is a category of an algorithm that allows software applications to become more accurate in predicting outcomes without being explicitly programmed. The basic premise of machine learning is to build algorithms that can receive input data and use statistical analysis to predict an output while updating outputs as new data becomes available.


Types of Machine Learning
Machine Learning
Supervised
Unsupervised
Reinforcement
Task Driven (Predict next value)
Data Driven (Identify Clusters)
Learn from Mistakes

## 1.3 Introduction to NLP

NLP is a branch of Machine Learning that consists of systematic processes for analyzing, understanding, and deriving information from the text data in a smart and efficient manner. By utilizing NLP and its components, one can organize the massive chunks of text data, perform numerous automated tasks and solve a wide range of problems such as – automatic summarization, machine translation, named entity recognition, relationship extraction, sentiment analysis, speech recognition, and topic segmentation etc.

Before moving further, I would like to explain some terms that are used in the article:

- Tokenization – process of converting a text into tokens
- Tokens – words or entities present in the text
- Text object – a sentence or a phrase or a word or an article

# CHAPTER 2
# LITERATURE SURVEY

**Title** : Web Application for Screening Resume

**Author** : Nikita Jayakar; Sonia Sunny; Pheba Babu; M. Kiruthika; Ambarish Gurjar
**Publisher** : IEEE
**Year** : 2019
**Context** :

This paper focuses majorly on the design of the web application which will be used to screen resumes (Curriculum Vitae) for a particular job posting. In the proposed system, a web application will encourage the job applicant candidates as well as the recruiters to use it for job applications and screening of resumes. Recruitment is a tedious process wherein the first task for any recruiter is to screen the resumes. The proposed web application is designed in such a way that job applicants as well as recruiters can use it with ease for applying for job openings and screening respectively. The recruiters from various companies can post the details of the job openings available in their respective companies. The interactive web application will allow the job applicants to submit their resume and apply for their job postings they may still be interested in. The resumes submitted by the candidates are then compared with the job profile requirement posted by the company recruiter by using techniques like machine learning and Natural Language Processing (NLP). Scores can then be given to the resumes and they can be ranked from highest match to lowest match. This ranking is made visible only to the company recruiter who is interested in selecting the best candidates from a large pool of candidates.

**Title** : Based on the application of AI technology in resume analysis and job recommendation

**Author** : Taipei, Taiwan ROC ; Han-Yen Yu
**Publisher** : IEEE
**Year** : 2020
**Context** :

This study adopted machine learning- and text mining technology-based artificial intelligence and current big data technology to analyze the trendiness of online discussion. Developing a system that can be applied in large job fairs, where numerous job applicants seek to match with the maximum of job vacancies provided by companies possible. The developed system conducts personal competitiveness analysis, personality trait analysis, and gives job vacancy recommendations according to the electronic resumes job applicants submit. In addition, the system generates a talent recommendation list for the companies. The experimental results verified that the job vacancies recommended by the developed system desirably met job applicants’ expectations.

**Title** : Automatic Extraction of Segments from Resumes using Machine Learning

**Author** : B Gunaseelan; Supriya Mandal; V Rajagopalan
**Publisher** : IEEE
**Year** : 2020
**Context** :

Online recruitment systems or automatic resume processing systems are becoming more popular because it saves time for both employers and job seekers. Manually processing these resumes and fitting to several job specifications is a difficult task. Due to the increased amount of data, it is a big challenge to effectively analyze each resume based on various parameters like experience, skill set, etc. Processing, extracting information and reviewing these applications automatically would save time and money. Automatic data extraction, focused primarily on skillset, experience and education from a resume. So, it is extremely helpful to map the appropriate resume for the right job description. In this research study, we propose a system that uses multi-level classification techniques to automatically extract detailed segment information like skillset, experience and education from a resume based on specific parameters. We have achieved state-of-the-art accuracy in the segment of the resumes to identify skill sets.

**Title** : Resume Scanning and Emotion Recognition System based on Machine Learning Algorithms

**Author** : R G Vishruth PES University, Bangalore, Karnataka, India ; R Sunitha; K S Varuna; N Varshini; Prasad B Honnavalli
**Publisher** : IEEE
**Year** : 2020
**Context** :

In the current smart world, everything should be done faster, smarter, and accurate way. The various organization's recruitment processes will be done face to face in an arranged venue. But, during some pandemics like Covid-19 face to face recruitment process will be very difficult. In the proposed system, a smarter way of performing the recruitment processes anywhere around the world based on the company requirements is performed. The aim of this article deals with making the process of candidate recruitment easier for companies. The amount of manual work that goes into recruiting processes is reduced and the initial scanning process of candidates was performed. By eliminating the redundant candidates helps in retaining only the applicable ones. Achieve this through the help of resume scanning, initial aptitude testing of candidates, and an interview session where the candidate answers questions asked by the interviewer. With this model, all the time and manual labor that is wasted in eliminating the redundant candidates is accomplished. It chooses the one who is best applicable to a job by comparing it with the job description based on the resumes received. Our model is working accurately for some of the predefined parameters of the company in a recruitment process by providing more security and reliability.

**Title** : Hybrid Job and Resume Matcher

**Author** : Nimet Tülümen; Gökhan Akgün; Ali Nohutçu; Günnur Sevgi Aktoros Genç; Serdar Gen
**Publisher** : IEEE
**Year** : 2021
**Context** :
Information extraction from text data has always been a tricky and difficult task. This work follows a previous work regarding a designed system that matches job ads with resumes, then assigns them a scoring point. In this system, there are two main parts: information extraction and scoring. For the information extraction part, rule-based methods are efficient when the format of the resumes and job ads are known. In this paper, powerful and efficient methods for information extraction from the mixed resume format and job ads using machine learning and deep learning methods are proposed.

# Chapter 3
# System Analysis

## 3.1 EXISTING SYSTEM

The existing system used the traditional way of machine learning algorithm and basics of NLP to pre process the text which is again fed into the machine learning classifier model. It can just classify whether the resume is good for the specific job role. And the system is only able to do only one specific job role and it's fixed at a time and it's not flexible for different domains of job role.

## 3.2 Disadvantages

Since the existing system is just using the existing job role and quite not suitable for real time high level resume screening for different different domain aspects. The existing system is time consuming and not cost efficient since it has to maintain a database regularly to store resumes and do all the pre processing.

## 3.3 Proposed system

Our proposed system uses a high level web application using streamlit which is used to classify the resume in a real time dynamic job profile. Our model can be deployed to any system which meets the basic requirements and can be able to classify even 1000s and 1000s of resumes with different job profiles and all are done instantly in real time mannar. With this, we can also see how the resume is about using word clouds and analyze the resume in real time using an interactive dashboard.

# CHAPTER 4
# REQUIREMENT SPECIFICATION

## 4.1 SOFTWARE REQUIREMENTS:

- Operating System - Ubuntu / Windows / Mac
- Language - Python 3.6 and above
- Updated Browser - Chrome / Firefox
- Streamlit

## 4.2 HARDWARE REQUIREMENTS:

- Processor - Intel Core i3/i5/i7 or AMD
- Hard Disk - Minimum (500GB)
- Ram - Minimum (4GB )

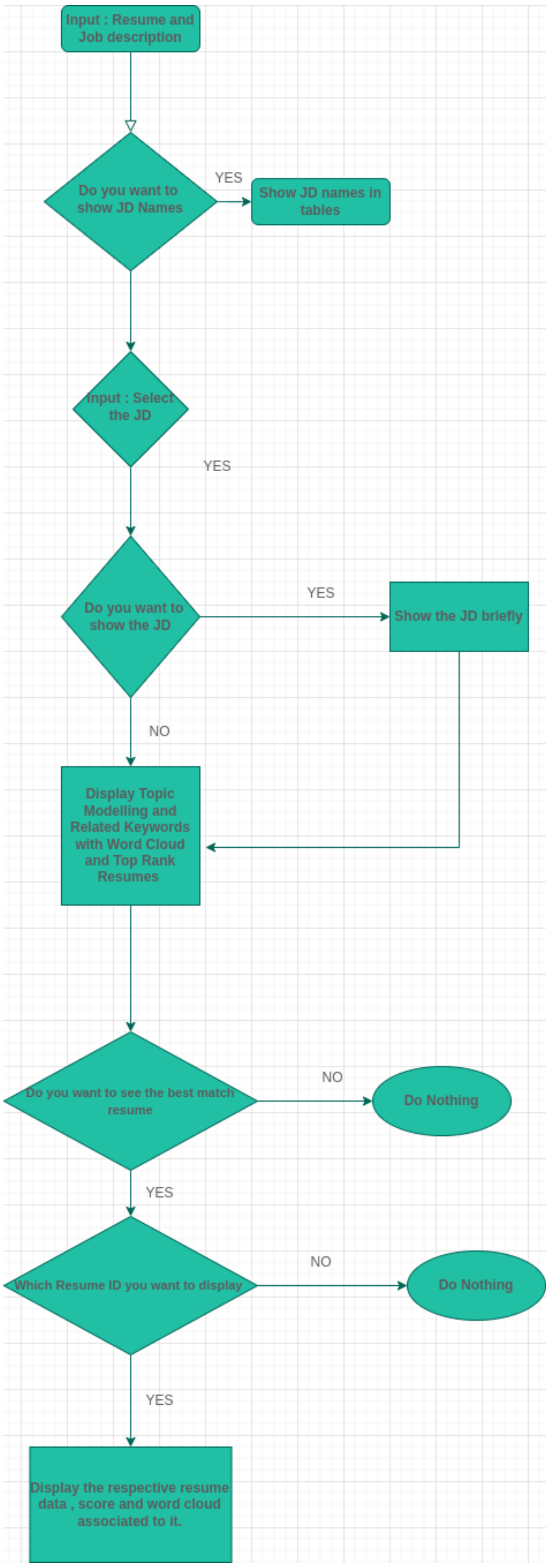
# CHAPTER - 05
# SYSTEM DESIGN

## 5.1 UML

Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects. Design is a meaningful engineering representation of a thing that is to be built.

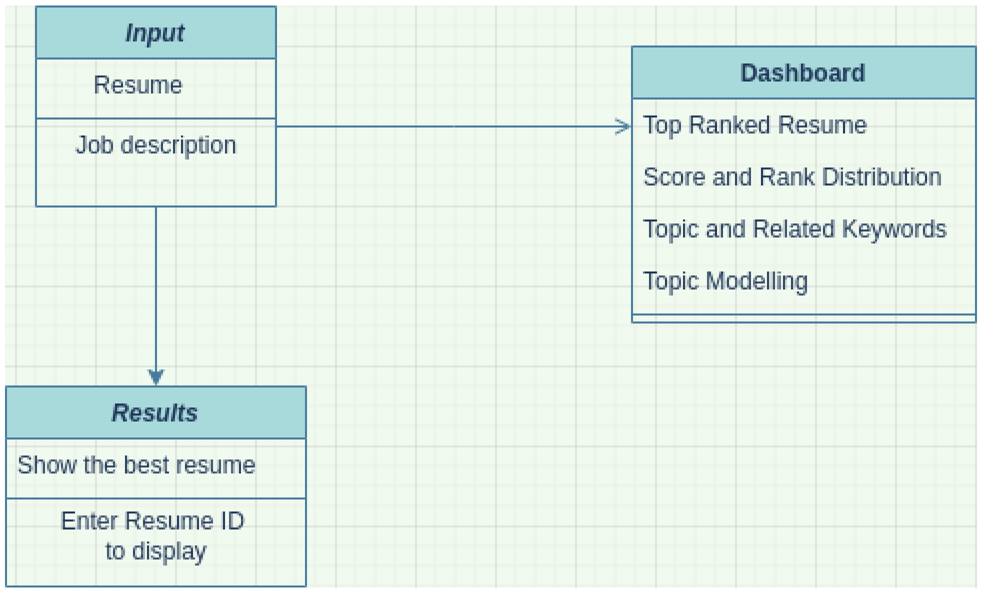
## 5.2 USE CASE

Use case diagrams are done in an early phase of a software development project. They express how it should be possible to use the final system. Use case diagrams are behavior diagrams used to describe a set of actions(use cases) that some system or systems (subject) should or can perform in collaboration with
one or more external users of the system (actors).


Input : Resume and Job description
Do you want to show JD Names
YES
Show JD names in tables
Input : Select the JD
YES
Do you want to show the JD
YES
Show the JD briefly
NO
Display Topic Modelling and Related Keywords with Word Cloud and Top Rank Resumes
Do you want to see the best match resume
NO
Do Nothing
YES
Which Resume ID you want to display
NO
Do Nothing
YES
Display the respective resume data , score and word cloud associated to it.

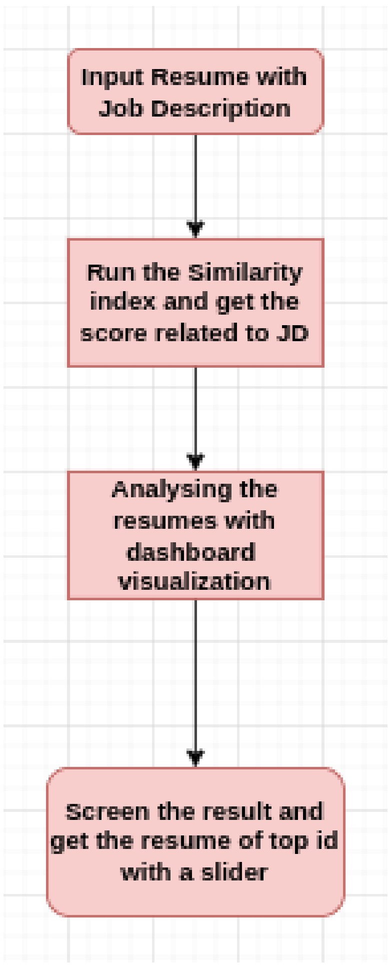
## 5.3 CLASS DIAGRAM

Class diagrams describe systems by illustrating attributes, operations and relationships between classes. Unified Modeling Language (UML) calls them structure diagrams. They work according to the principles of object orientation. This orientation describes how objects interact with each other. Class diagrams
give you the ability to create models with the help of UML using attributes, relationships, operations and intersections



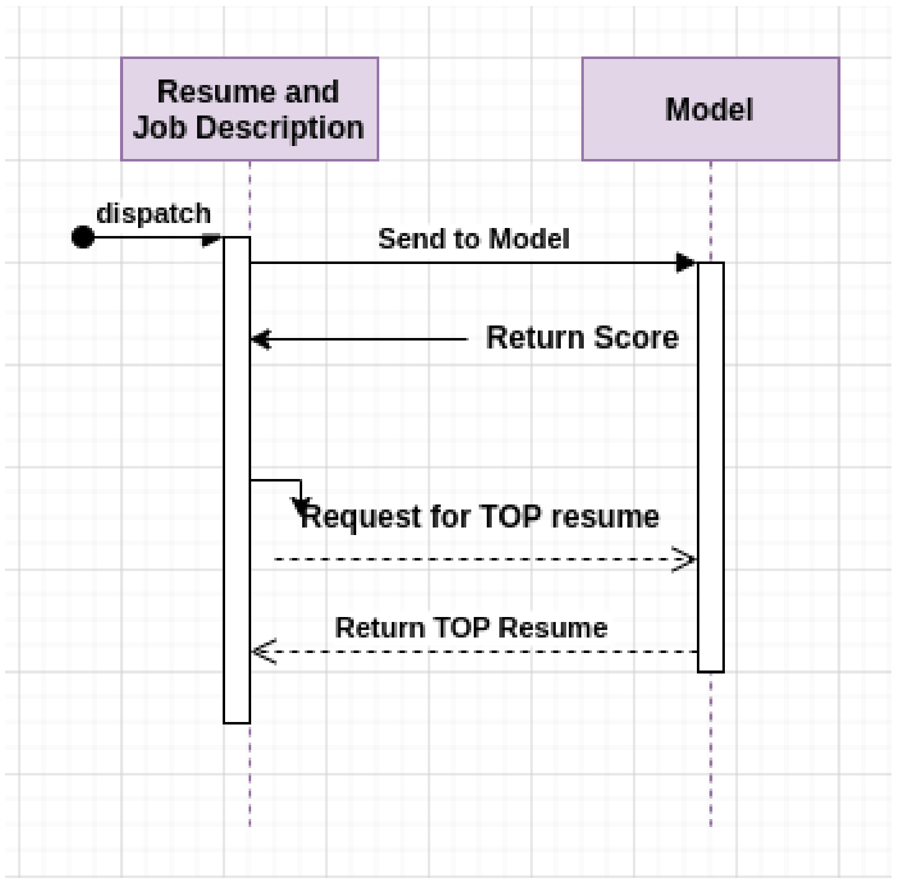
## 5.4 ACTIVITY DIAGRAM

Activity diagrams show the state of activities by showing the sequence of activities performed. Activity diagrams describe the workflow behavior of a system.The basic purpose of activity diagrams is similar to other four diagrams. It captures the dynamic behavior of the system. Other four diagrams
are used to show the message flow from one object to another but the activity diagram is used to show message flow from one activity to another.


Input Resume with Job Description
Run the Similarity index and get the score related to JD
Analysing the resumes with dashboard visualization
Screen the result and get the resume of top id with a slider

## 5.5 SEQUENCE DIAGRAM

Sequence Diagrams are interaction diagrams that detail how operations are carried out. They capture the interaction between objects in the context of a collaboration. Sequence Diagrams are time focused and they show the order of the interaction visually by using the vertical axis of the diagram to represent time, what messages are sent and when.



# CHAPTER 6
# IMPLEMENTATION OF MODULES MODULES

The proposed system consist of 3 modules

1) Importing and Pre processing resume & Job description
2) Score calculating using multi similarity index and model building with Gensim LDA and TF-IDF
3) Creating Interactive Web application Dashboard using streamlit

## 6.1 Module 1 : Importing and Pre processing resume & Job description

Resume & Job description cleaning and pre processing using NLP , Term frequency and inverse doc frequency

## 6.2 Module 2 : Score calculating using multi similarity index and model building with Gensim LDA and TF-IDF

Creating function to calculate the score for each resume with respective to job description and building word cloud with LDA, Different similarity index

## 6.3 Module 3 : Creating Interactive Web application Dashboard using streamlit

Creating interactive and real time streamlit web application for interactive dashboard for job description, resume content, sliders, top resume , wordcloud

## Module 1

### Importing and Pre processing resume & Job description

### Natural Language Processing

Since, text is the most unstructured form of all the available data, various types of noise are present in it and the data is not readily analyzable without any pre-processing. The entire process of cleaning and standardization of text, making it noise-free and ready for analysis is known as text preprocessing.It is predominantly comprised of three steps:

- Noise Removal
- Lexicon Normalization
- Object Standardization

**Noise Removal**

Any piece of text which is not relevant to the context of the data and the end-output can be specified as the noise.

For example – language stopwords (commonly used words of a language – is, am, the, of, in etc), URLs or links, social media entities (mentions, hashtags), punctuations and industry specific words. This step deals with removal of all types of noisy entities present in the text.

A general approach for noise removal is to prepare a dictionary of noisy entities, and iterate the text object by tokens (or by words), eliminating those tokens which are present in the noise dictionary.

**Lexicon Normalization**

Another type of textual noise is about the multiple representations exhibited by a single word.

For example – “play”, “player”, “played”, “plays” and “playing” are the different variations of the word – “play”, Though they mean different but contextually all are similar. The step converts all the disparities of a word into their normalized form (also known as lemma). Normalization is a pivotal step for feature engineering with text as it converts the high dimensional features (N different features) to the low dimensional space (1 feature), which is an ideal ask for any ML model.

The most common lexicon normalization practices are :

- Stemming: Stemming is a rudimentary rule-based process of stripping the suffixes (“ing”, “ly”, “es”, “s” etc) from a word.
- Lemmatization: Lemmatization, on the other hand, is an organized & step by step procedure of obtaining the root form of the word, it makes use of vocabulary (dictionary importance of words) and morphological analysis (word structure and grammar relations).

## Object Standardization

Text data often contains words or phrases which are not present in any standard lexical dictionaries. These pieces are not recognized by search engines and models.Some of the examples are – acronyms, hashtags with attached words, and colloquial slangs. With the help of regular expressions and manually prepared data dictionaries, this type of noise can be fixed

## MODULE - 02

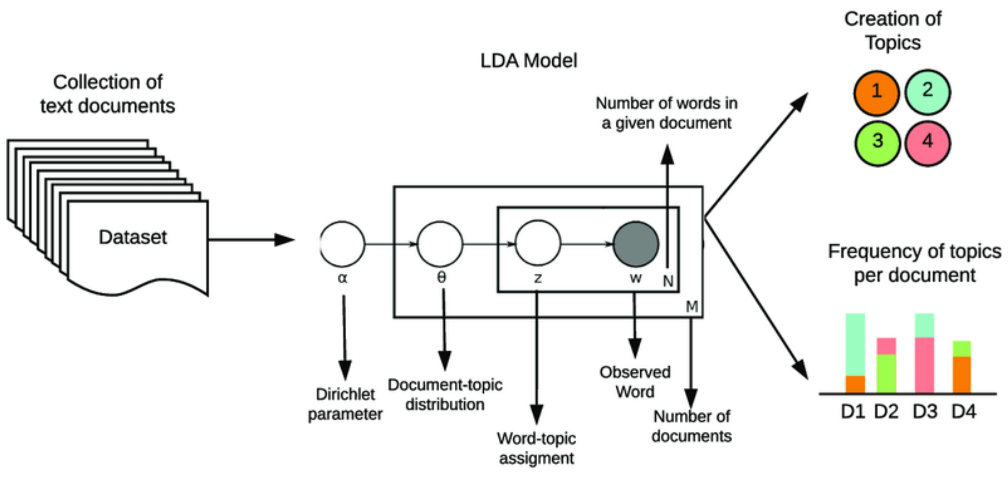
### Score calculating using multi similarity index and model building with Gensim LDA and TF-IDF

## Topic Modeling

Topic modeling is a process of automatically identifying the topics present in a text corpus; it derives the hidden patterns among the words in the corpus in an unsupervised manner. Topics are defined as “a repeating pattern of co-occurring terms in a corpus”. A good topic model results in – “health”, “doctor”, “patient”, “hospital” for a topic – Healthcare, and “farm”, “crops”, “wheat” for a topic – “Farming”.

### Latent Dirichlet Allocation (LDA)

Latent Dirichlet Allocation (LDA) is the most popular topic modelling technique, Following is the code to implement topic modeling using LDA in python



LDA assumes documents are produced from a mixture of topics. Those topics then generate words based on their probability distribution. Given a dataset of documents, LDA backtracks and tries to figure out what topics would create those documents in the first place.

LDA model looks for repeating term patterns in the entire DT matrix. Python provides many great libraries for text mining practices, "gensim" is one such clean and beautiful library to handle text data. It is scalable, robust and efficient. Following code shows how to convert a corpus into a document-term matrix.

LDA is a matrix factorization technique. In vector space, any corpus (collection of documents) can be represented as a document-term matrix. The following matrix shows a corpus of N documents D1, D2, D3 … Dn and vocabulary size of M words W1,W2 .. Wn. The value of i,j cell gives the frequency count of word Wj in Document Di.

|  | W1 | W2 | W3 | Wn |
|---|---|---|---|---|
| D1 | 0 | 2 | 1 | 3 |
| D2 | 1 | 4 | 0 | 0 |
| D3 | 0 | 2 | 3 | 1 |
| Dn | 1 | 1 | 3 | 0 |

LDA converts this Document-Term Matrix into two lower dimensional matrices – M1 and M2.M1 is a document-topics matrix and M2 is a topic – terms matrix with dimensions (N, K) and (K, M) respectively, where N is the number of documents, K is the number of topics and M is the vocabulary size.

|  | K1 | K2 | K3 | K |
|---|---|---|---|---|
| D1 | 1 | 0 | 0 | 1 |
| D2 | 1 | 1 | 0 | 0 |
| D3 | 1 | 0 | 0 | 1 |
| Dn | 1 | 0 | 1 | 0 |

|  | W1 | W2 | W3 | Wm |
|---|---|---|---|---|
| K1 | 0 | 1 | 1 | 1 |
| K2 | 1 | 1 | 1 | 0 |
| K3 | 1 | 0 | 0 | 1 |
| K | 1 | 1 | 0 | 0 |

Notice that these two matrices already provide topic word and document topic distributions, However, these distributions need to be improved, which is the main aim of LDA. LDA makes use of sampling techniques in order to improve these matrices.

It Iterates through each word “w” for each document “d” and tries to adjust the current topic – word assignment with a new assignment. A new topic “k” is assigned to word “w” with a probability P which is a product of two probabilities p1 and p2.

For every topic, two probabilities p1 and p2 are calculated. P1 – p(topic t / document d) = the proportion of words in document d that are currently assigned to topic t. P2 – p(word w / topic t) = the proportion of assignments to topic t over all documents that come from this word w.

The current topic – word assignment is updated with a new topic with the probability, product of p1 and p2 . In this step, the model assumes that all the existing word – topic assignments except the current word are correct. This is essentially the probability that topic t generated word w, so it makes sense to adjust the current word’s topic with new probability.

After a number of iterations, a steady state is achieved where the document topic and topic term distributions are fairly good. This is the convergence point of LDA.

## Parameters of LDA

**Alpha and Beta Hyperparameters** – alpha represents document-topic density and Beta represents topic-word density. Higher the value of alpha, documents are composed of more topics and lower the value of alpha, documents contain fewer topics. On the other hand, higher the beta, topics are composed of a large number of words in the corpus, and with the lower value of beta, they are composed of few words.

**Number of Topics** – Number of topics to be extracted from the corpus. Researchers have developed approaches to obtain an optimal number of topics by using Kullback Leibler Divergence Score. I will not discuss this in detail, as it is too mathematical. For understanding, one can refer to this[1] original paper on the use of KL divergence.

Number of Topic Terms – Number of terms composed in a single topic. It is generally decided according to the requirement. If the problem statement talks about extracting themes or concepts, it is recommended to choose a higher number, if the problem statement talks about extracting features or terms, a low number is recommended.

**Number of Iterations / passes** – Maximum number of iterations allowed to LDA algorithm for convergence.

## Term Frequency – Inverse Document Frequency (TF – IDF)

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

**TF-IDF**

Term $x$ within document $y$

$tf_{x,y}$ = frequency of $x$ in $y$

$df_x$ = number of documents containing $x$

$N$ = total number of documents

TF-IDF is a weighted model commonly used for information retrieval problems. It aims to convert the text documents into vector models on the basis of occurrence of words in the documents without taking into consideration the exact ordering. For Example – let say there is a dataset of N text documents, In any document “D”, TF and IDF will be defined as –

Term Frequency (TF) – TF for a term “t” is defined as the count of a term “t” in a document “D”

Inverse Document Frequency (IDF) – IDF for a term is defined as the logarithm of ratio of total documents available in the corpus and number of documents containing the term T.

TF . IDF – TF IDF formula gives the relative importance of a term in a corpus (list of documents), given by the following formula below. Following is the code using python’s scikit learn package to convert a text into tf idf vectors:

## Count / Density / Readability Features

Count or Density based features can also be used in models and analysis. These features might seem trivial but show a great impact in learning models. Some of the features are: Word Count, Sentence Count, Punctuation Counts and Industry specific word counts. Other types of measures include readability measures such as syllable counts, smog index and flesch reading ease. Refer to Textstat library to create such features.

**Word Embedding (text vectors)**

Word embedding is the modern way of representing words as vectors. The aim of word embedding is to redefine the high dimensional word features into low dimensional feature vectors by preserving the contextual similarity in the corpus. They are widely used in deep learning models such as Convolutional Neural Networks and Recurrent Neural Networks.

Word2Vec and GloVe are the two popular models to create word embedding of a text. These models take a text corpus as input and produce the word vectors as output.

Word2Vec model is composed of a preprocessing module, a shallow neural network model called Continuous Bag of Words and another shallow neural network model called skip-gram. These models are widely used for all other nlp problems. It first constructs a vocabulary from the training corpus and then learns word embedding representations.

They can be used as feature vectors for ML models, used to measure text similarity using cosine similarity techniques, words clustering and text classification techniques.

## Text Matching / Similarity

```
    j = td.jaccard.similarity(resume, job_des)
    s = td.sorensen_dice.similarity(resume, job_des)
    c = td.cosine.similarity(resume, job_des)
    o = td.overlap.normalized_similarity(resume, job_des)
```

One of the important areas of NLP is the matching of text objects to find similarities. Important applications of text matching include automatic spelling correction, data de-duplication and genome analysis etc.

A number of text matching techniques are available depending upon the requirement.

**Jaccard index**– Falling under the set similarity domain, the formula is to find the number of common tokens and divide it by the total number of unique tokens. Its expressed in the mathematical terms by,

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

where, the numerator is the intersection (common tokens) and denominator is union (unique tokens). The second case is for when there is some overlap, for which we must remove the common terms as they would add up twice by combining all tokens of both strings. As the required input is tokens instead of complete strings, it falls to the user to efficiently and intelligently tokenize his string, depending on the use case.

**Sorensen-Dice**

Falling under set similarity, the logic is to find the common tokens, and divide it by the total number of tokens present by combining both sets. The formulae is

$$DSC = \frac{2|X \cap Y|}{|X| + |Y|}$$

where, the numerator is twice the intersection of two sets/strings. The idea behind this is if a token is present in both strings, its total count is obviously twice the intersection (which removes duplicates). The denominator is a simple combination of all tokens in both strings. Note, it's quite different from the jaccard’s denominator, which was a union of two strings. As the case with intersection, union too removes duplicates and this is avoided in dice algorithm. Because of this, dice will always overestimate the similarity between two strings.

**Cosine Similarity** – When the text is represented as vector notation, a general cosine similarity can also be applied in order to measure vectorized similarity. Following code converts a text to vectors (using term frequency) and applies cosine similarity to provide closeness among two texts.

$$\text{cosine similarity} = S_C(A, B) := \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2} \sqrt{\sum\limits_{i=1}^{n} B_i^2}},$$

**Overlap Similarity-**

The **overlap**,[1] or **Szymkiewicz–Simpson coefficient**, is a similarity measure that measures the overlap between two finite sets. It is related to the Jaccard index and is defined as the size of the intersection divided by the smaller of the size of the two sets:

$$\text{overlap}(X,Y) = \frac{|X \cap Y|}{\min(|X|,|Y|)}$$

## SKLEARN

**Scikit-learn** (formerly **scikits.learn** and also known as **sklearn**) is a free software machine learning library for the Python programming language

**Scikit-learn** or **Sklearn** is the **open-source python library** useful in machine learning for predictive data analysis.

It is built on **NumPy**, **SciPy** and **Matplotlib**, that means, it has library such as NumPy, SciPy and many more which makes our work easier with arrays and machine learning techniques.

It comes with a number of tools for data preprocessing, model selection and model evaluation including regression, classification, clustering and dimensionality reduction.

## NLTK

Natural language processing (NLP) is a field that focuses on making natural human language usable by computer programs. NLTK, or Natural Language Toolkit, is a Python package that you can use for NLP.

A lot of the data that you could be analyzing is unstructured data and contains human-readable text. Before you can analyze that data programmatically, you first need to preprocess it.

## NLTK (Natural Language Toolkit)

Library is a suite that contains libraries and programs for statistical language processing. It is one of the most powerful NLP libraries, which contains packages to make machines understand human language and reply to it with an appropriate response.

## SPACY

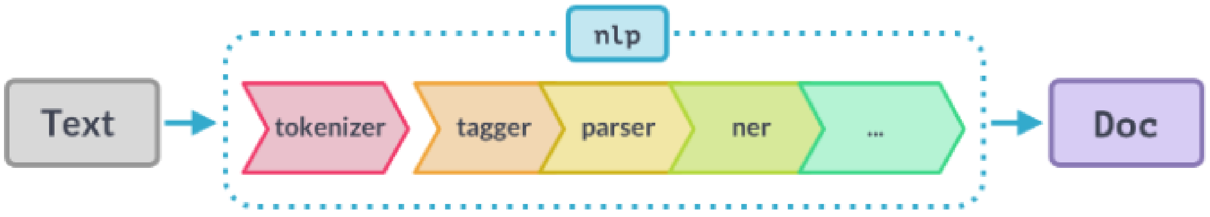
SpaCy's Statistical Models.These models are the power engines of spaCy. These models enable spaCy to perform several NLP related tasks, such as part-of-speech tagging, named entity recognition, and dependency parsing.

- en_core_web_sm: English multi-task CNN trained on OntoNotes. Size – 11 MB

- en_core_web_md: English multi-task CNN trained on OntoNotes, with GloVe vectors trained on Common Crawl. Size – 91 MB
- en_core_web_lg: English multi-task CNN trained on OntoNotes, with GloVe vectors trained on Common Crawl. Size – 789 MB

## SpaCy's Processing Pipeline

The first step for a text string, when working with spaCy, is to pass it to an NLP object. This object is essentially a pipeline of several text pre-processing operations through which the input text string has to go through.



*Source: https://course.spacy.io/chapter3*

As you can see in the figure above, the NLP pipeline has multiple components, such as *tokenizer*, *tagger*, *parser*, *ner*, etc. So, the input text string has to go through all these components before we can work on it.

## Gensim

It is billed as a Natural Language Processing package that does 'Topic Modeling for Humans'. But it is practically much more than that. It is a leading and a state-of-the-art package for processing texts, working with word vector models (such as Word2Vec, FastText etc) and for building topic models.

In order to work on text documents, Gensim requires the words (aka tokens) be converted to unique ids. In order to achieve that, Gensim lets you create a `Dictionary` object that maps each word to a unique id. So, how to create a `Dictionary`? By converting your text/sentences to a [list of words] and passing it to the `corpora.Dictionary()` object. We will see how to actually do this in the next section. But why is the dictionary object needed and where can it be used? The dictionary object is typically used to create a 'bag of words' Corpus. It is this Dictionary and the bag-of-words (Corpus) that are used as inputs to topic modeling and other models that Gensim specializes in. Alright, what sort of text inputs can gensim handle? The input text typically comes in 3 different forms:

1. As sentences stored in python’s native list object
2. As one single text file, small or large.
3. In multiple text files.

Now, when your text input is large, you need to be able to create the dictionary object without having to load the entire text file. The good news is Gensim lets you read the text and update the dictionary, one line at a time, without loading the entire text file into system memory. Let’s see how to do that in the next 2 sections. But, before we get in, let’s understand some NLP jargon. A ‘token’ typically means a ‘word’. A ‘document’ can typically refer to a ‘sentence’ or ‘paragraph’ and a ‘corpus’ is typically a ‘collection of documents as a bag of words’. That is, for each document, a corpus contains each word’s id and its frequency count in that document. As a result, information about the order of words is lost.

## Module - 03 : Creating Interactive Web application Dashboard using streamlit

### STREAMLIT

Streamlit is an open-source python framework for building web apps for Machine Learning and Data Science. We can instantly develop web apps and deploy them easily using Streamlit. Streamlit allows you to write an app the same way you write a python code. Streamlit makes it seamless to work on the interactive loop of coding and viewing results in the web app.

## Installing Streamlit

1. Make sure you have python installed in your system

2. Use the following command to install streamlit,

```
pip install streamlit
```

### Running a streamlit app

First, we create a python script with streamlit commands and execute the script using the following command,

```
streamlit run <yourscript.py>
```

## Development flow

Every time you want to update your app, save the source file. When you do that, Streamlit detects if there is a change and asks you whether you want to rerun your app. Choose "Always rerun" at the top-right of your screen to automatically update your app every time you change its source code.

This allows you to work in a fast interactive loop: you type some code, save it, try it out live, then type some more code, save it, try it out, and so on until you're happy with the results. This tight loop between coding and viewing results live is one of the ways Streamlit makes your life easier.

## Data flow

Streamlit allows you to write an app the same way you write a python code. The streamlit has a distinctive data flow, any time something changes in your code or anything needs to be updated on the screen, streamlit reruns your python script entirely from the top to the bottom. This happens when the user interacts with the widgets like a select box or drop-down box or when the source code is changed.

If you have some costly operations while rerunning your web app, like loading data from databases, you can use streamlit’s st.cache method to cache those datasets, so that it loads faster.

## Displaying the data

Streamlit provides you with many methods to display various types of data like arrays, tables, and data frames.

- To write a string simply use, st.write(“Your string”)
- To display a data frame use, st.dataframe method

## Widgets

There are several widgets available in streamlit, like st.selectbox, st.checkbox, st.slider, and etc. Let us see an example for widget in streamlit.

```
import streamlit as st

value = st.slider('val')  # this is a widget

st.write(value, 'squared is', value * value)
```

During the first run, the web app would output the string “0 squared is 0”, and then if a user increases or decreases the widget, the code is rerun by the streamlit from top to bottom and assigns the present state of the widget to the variable.

## Layout

You can easily arrange your widgets or data seamlessly using the st.sidebar method. This method helps you to align data in the left panel sidebar. All you have to do is simply use st.sidebar.selectbox to display a selectbox in the left panel.

Select box in streamlit

```
import streamlit as st

st.title("Welcome to Streamlit!")

selectbox = st.selectbox(

    "Select yes or no",

    ["Yes", "No"]

)

st.write(f"You selected {selectbox}")
```

The first argument to st.selectbox is the string to display and the second argument is a list of options to select. And then we display the selected value in the app using st.write method.

Run the following command to see the results,

```
streamlit run selectbox.py
```

# Welcome to Streamlit!

Select yes or no

Yes

You selected Yes

# CHAPTER - 07
# SOURCE CODE

## 7.1 APP.PY

```
import matplotlib.colors as mcolors
import gensim
import gensim.corpora as corpora
from operator import index
from wordcloud import WordCloud
from pandas._config.config import options
import pandas as pd
import streamlit as st
import plotly.express as px
import plotly.graph_objects as go
import matplotlib.pyplot as plt
import Similar
from PIL import Image
import time


image = Image.open('Images//logo.png')
st.image(image, use_column_width=True)

st.title("Resume Matcher")


# Reading the CSV files prepared by the fileReader.py
Resumes = pd.read_csv('Resume_Data.csv')
```

```
Jobs = pd.read_csv('Job_Data.csv')


################################ JOB DESCRIPTION CODE
#######################################
# Checking for Multiple Job Descriptions
# If more than one Job Descriptions are available, it
asks user to select one as well.
if len(Jobs['Name']) <= 1:
   st.write(
       "There is only 1 Job Description present. It
will be used to create scores.")
else:
   st.write("There are ", len(Jobs['Name']),
            "Job Descriptions available. Please
select one.")


# Asking to Print the Job Desciption Names
if len(Jobs['Name']) > 1:
   option_yn = st.selectbox(
       "Show the Job Description Names?",
options=['YES', 'NO'])
   if option_yn == 'YES':
       index = [a for a in range(len(Jobs['Name']))]
       fig =
go.Figure(data=[go.Table(header=dict(values=["Job
No.", "Job Desc. Name"], line_color='darkslategray',
fill_color='lightskyblue'),

cells=dict(values=[index, Jobs['Name']],
line_color='darkslategray',

fill_color='cyan'))
                             ])
       fig.update_layout(width=700, height=400)
       st.write(fig)



# Asking to chose the Job Description
index = st.slider("Which JD to select ? : ", 0,
                 len(Jobs['Name'])-1, 1)



option_yn = st.selectbox("Show the Job Description
?", options=['YES', 'NO'])
if option_yn == 'YES':
   st.markdown("---")
   st.markdown("### Job Description :")
   fig = go.Figure(data=[go.Table(
       header=dict(values=["Job Description"],
                   fill_color='#f0a500',
                   align='center',
font=dict(color='white', size=16)),
       cells=dict(values=[Jobs['Context'][index]],
                  fill_color='#f4f4f4',
                  align='left'))])

   fig.update_layout(width=800, height=500)
   st.write(fig)
   st.markdown("---")


#################################### SCORE CALCUATION
################################
@st.cache()
def calculate_scores(resumes, job_description):
   scores = []
   for x in range(resumes.shape[0]):
       score = Similar.match(
           resumes['TF_Based'][x],
job_description['TF_Based'][index])
       scores.append(score)
   return scores


Resumes['Scores'] = calculate_scores(Resumes, Jobs)

Ranked_resumes = Resumes.sort_values(
   by=['Scores'],
ascending=False).reset_index(drop=True)

Ranked_resumes['Rank'] = pd.DataFrame(
   [i for i in range(1,
len(Ranked_resumes['Scores'])+1)])
```

```
######################################## SCORE TABLE
PLOT ######################################

fig1 = go.Figure(data=[go.Table(
   header=dict(values=["Rank", "Name", "Scores"],
               fill_color='#00416d',
               align='center',
font=dict(color='white', size=16)),
   cells=dict(values=[Ranked_resumes.Rank,
Ranked_resumes.Name, Ranked_resumes.Scores],
              fill_color='#d6e0f0',
              align='left'))])

fig1.update_layout(title="Top Ranked Resumes",
width=700, height=1100)
st.write(fig1)

st.markdown("---")

fig2 = px.bar(Ranked_resumes,
             x=Ranked_resumes['Name'],
y=Ranked_resumes['Scores'], color='Scores',
             color_continuous_scale='haline',
title="Score and Rank Distribution")
# fig.update_layout(width=700, height=700)
st.write(fig2)
st.markdown("---")

############################################ TF-IDF
Code ###################################


@st.cache()
def get_list_of_words(document):
   Document = []

   for a in document:
       raw = a.split(" ")
       Document.append(raw)

   return Document


document = get_list_of_words(Resumes['Cleaned'])

id2word = corpora.Dictionary(document)
corpus = [id2word.doc2bow(text) for text in document]


lda_model =
gensim.models.ldamodel.LdaModel(corpus=corpus,
id2word=id2word, num_topics=6, random_state=100,

update_every=3, chunksize=100, passes=50,
alpha='auto', per_word_topics=True)
```

```
################################### LDA CODE
##############################################


@st.cache  # Trying to improve performance by
reducing the rerun computations
def format_topics_sentences(ldamodel, corpus):
   sent_topics_df = []
   for i, row_list in enumerate(ldamodel[corpus]):
       row = row_list[0] if ldamodel.per_word_topics
else row_list
       row = sorted(row, key=lambda x: (x[1]),
reverse=True)
       for j, (topic_num, prop_topic) in
enumerate(row):
           if j == 0:
               wp = ldamodel.show_topic(topic_num)
               topic_keywords = ", ".join([word for
word, prop in wp])
               sent_topics_df.append(
                   [i, int(topic_num),
round(prop_topic, 4)*100, topic_keywords])
           else:
               break

   return sent_topics_df
```

```
################################### Topic Word Cloud
Code ########################################
# st.sidebar.button('Hit Me')
st.markdown("## Topics and Topic Related Keywords ")
st.markdown(
   """This Wordcloud representation shows the Topic
Number and the Top Keywords that contstitute a Topic.
   This further is used to cluster the resumes.
""")

cols = [color for name, color in
mcolors.TABLEAU_COLORS.items()]

cloud = WordCloud(background_color='white',
                 width=2500,
                 height=1800,
                 max_words=10,
                 colormap='tab10',
                 collocations=False,
                 color_func=lambda *args, **kwargs:
cols[i],
                 prefer_horizontal=1.0)

topics = lda_model.show_topics(formatted=False)

fig, axes = plt.subplots(2, 3, figsize=(10, 10),
sharex=True, sharey=True)

for i, ax in enumerate(axes.flatten()):
   fig.add_subplot(ax)
   topic_words = dict(topics[i][1])
   cloud.generate_from_frequencies(topic_words,
max_font_size=300)
   plt.gca().imshow(cloud)
   plt.gca().set_title('Topic ' + str(i),
fontdict=dict(size=16))
   plt.gca().axis('off')


plt.subplots_adjust(wspace=0, hspace=0)
plt.axis('off')
plt.margins(x=0, y=0)
plt.tight_layout()
st.pyplot(plt)

st.markdown("---")

####################### SETTING UP THE DATAFRAME FOR
SUNBURST-GRAPH ############################

df_topic_sents_keywords = format_topics_sentences(
   ldamodel=lda_model, corpus=corpus)
df_some = pd.DataFrame(df_topic_sents_keywords,
columns=[
                      'Document No', 'Dominant
Topic', 'Topic % Contribution', 'Keywords'])
df_some['Names'] = Resumes['Name']
```

```
df = df_some

st.markdown("## Topic Modelling of Resumes ")
st.markdown(
   "Using LDA to divide the topics into a number of
usefull topics and creating a Cluster of matching
topic resumes.  ")
fig3 = px.sunburst(df, path=['Dominant Topic',
'Names'], values='Topic % Contribution',
                  color='Dominant Topic',
color_continuous_scale='viridis', width=800,
height=800, title="Topic Distribution Graph")
st.write(fig3)


############################## RESUME PRINTING
#############################

option_2 = st.selectbox("Show the Best Matching
Resumes?", options=[
   'YES', 'NO'])
if option_2 == 'YES':
   indx = st.slider("Which resume to display ?:",
                    1, Ranked_resumes.shape[0], 1)

   st.write("Displaying Resume with Rank: ", indx)
   st.markdown("---")
   st.markdown("## **Resume** ")
   value = Ranked_resumes.iloc[indx-1, 2]
   st.markdown("#### The Word Cloud For the Resume")
   wordcloud = WordCloud(width=800, height=800,
                         background_color='white',
                         colormap='viridis',
collocations=False,

min_font_size=10).generate(value)
   plt.figure(figsize=(7, 7), facecolor=None)
   plt.imshow(wordcloud)
   plt.axis("off")
   plt.tight_layout(pad=0)
   st.pyplot(plt)

   st.write("With a Match Score of :",
Ranked_resumes.iloc[indx-1, 6])
   fig = go.Figure(data=[go.Table(
       header=dict(values=["Resume"],
                   fill_color='#f0a500',
                   align='center',
font=dict(color='white', size=16)),
       cells=dict(values=[str(value)],
                  fill_color='#f4f4f4',
                  align='left'))])

   fig.update_layout(width=800, height=1200)
   st.write(fig)
   # st.text(df_sorted.iloc[indx-1, 1])
   st.markdown("---")
```

## 7.2 FILE READER.PY

```
from operator import index
from pandas._config.config import options
import Cleaner
import textract as tx
import pandas as pd
import os
import tf_idf

resume_dir = "Data/Resumes/"
job_desc_dir = "Data/JobDesc/"
resume_names = os.listdir(resume_dir)
job_description_names = os.listdir(job_desc_dir)

document = []


def read_resumes(list_of_resumes, resume_directory):
   placeholder = []
   for res in list_of_resumes:
       temp = []
       temp.append(res)
       text = tx.process(resume_directory+res,
encoding='ascii')
       text = str(text, 'utf-8')
       temp.append(text)
       placeholder.append(temp)
   return placeholder


document = read_resumes(resume_names, resume_dir)
```

```
def get_cleaned_words(document):
   for i in range(len(document)):
       raw = Cleaner.Cleaner(document[i][1])
       document[i].append(" ".join(raw[0]))
       document[i].append(" ".join(raw[1]))
       document[i].append(" ".join(raw[2]))
       sentence = tf_idf.do_tfidf(document[i][3].split(" "))
       document[i].append(sentence)
   return document


Doc = get_cleaned_words(document)

Database = pd.DataFrame(document, columns=[
                       "Name", "Context", "Cleaned",
"Selective", "Selective_Reduced", "TF_Based"])

Database.to_csv("Resume_Data.csv", index=False)

# Database.to_json("Resume_Data.json", index=False)


def read_jobdescriptions(job_description_names, job_desc_dir):
   placeholder = []
   for tes in job_description_names:
       temp = []
       temp.append(tes)
       text = tx.process(job_desc_dir+tes, encoding='ascii')
       text = str(text, 'utf-8')
       temp.append(text)
       placeholder.append(temp)
   return placeholder
```

```
job_document = read_jobdescriptions(job_description_names,
job_desc_dir)

Jd = get_cleaned_words(job_document)

jd_database = pd.DataFrame(Jd, columns=[
                           "Name", "Context", "Cleaned",
"Selective", "Selective_Reduced", "TF_Based"])

jd_database.to_csv("Job_Data.csv", index=False)
```

CLEANER.PY

```
import spacy
import Distill

try:
   nlp = spacy.load('en_core_web_sm')

except ImportError:
   print("Spacy's English Language Modules aren't present \n
Install them by doing \n python -m spacy download
en_core_web_sm")



def _base_clean(text):
   """
   Takes in text read by the parser file and then does the
text cleaning.
   """
   text = Distill.tokenize(text)
   text = Distill.remove_stopwords(text)
   text = Distill.remove_tags(text)
```

```
    text = Distill.lemmatize(text)
    return text


def _reduce_redundancy(text):
    """
    Takes in text that has been cleaned by the _base_clean and
uses set to reduce the repeating words
    giving only a single word that is needed.
    """
    return list(set(text))


def _get_target_words(text):
    """
    Takes in text and uses Spacy Tags on it, to extract the
relevant Noun, Proper Noun words that contain words related to
tech and JD.

    """
    target = []
    sent = " ".join(text)
    doc = nlp(sent)
    for token in doc:
        if token.tag_ in ['NN', 'NNP']:
            target.append(token.text)
    return target




def Cleaner(text):
    sentence = []
    sentence_cleaned = _base_clean(text)
    sentence.append(sentence_cleaned)
    sentence_reduced = _reduce_redundancy(sentence_cleaned)
    sentence.append(sentence_reduced)
    sentence_targetted = _get_target_words(sentence_reduced)
    sentence.append(sentence_targetted)
    return sentence
```

7.3 DISTILL.py

```
import nltk
import spacy
import re

from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.corpus import stopwords

# Define english stopwords
stop_words = stopwords.words('english')

# load the spacy module and create a nlp object
# This need the spacy en module to be present on the system.
nlp = spacy.load('en_core_web_sm')
# proces to remove stopwords form a file, takes an optional_word
list
# for the words that are not present in the stop words but the user
wants them deleted.


def remove_stopwords(text, stopwords=stop_words,
optional_params=False, optional_words=[]):
   if optional_params:
       stopwords.append([a for a in optional_words])
   return [word for word in text if word not in stopwords]
```

```python
def tokenize(text):
    # Removes any useless punctuations from the text
    text = re.sub(r'[^\w\s]', '', text)
    return word_tokenize(text)


def lemmatize(text):
    # the input to this function is a list
    str_text = nlp(" ".join(text))
    lemmatized_text = []
    for word in str_text:
        lemmatized_text.append(word.lemma_)
    return lemmatized_text

# internal fuction, useless right now.


def _to_string(List):
    # the input parameter must be a list
    string = " "
    return string.join(List)


def remove_tags(text, postags=['PROPN', 'NOUN', 'ADJ', 'VERB',
'ADV']):

    filtered = []
    str_text = nlp(" ".join(text))
    for token in str_text:
        if token.pos_ in postags:
            filtered.append(token.text)
    return filtered
```

## 7.4 REQUIREMENTS.TXT

```
altair==4.1.0
argcomplete==1.10.0
argon2-cffi==20.1.0
asgiref==3.4.1
astor==0.8.1
async-generator==1.10
attrs==21.2.0
autopep8==1.5.7
backcall==0.2.0
base58==2.1.0
beautifulsoup4==4.8.0
bleach==3.3.1
blinker==1.4
blis==0.7.4
cachetools==4.2.2
catalogue==2.0.4
certifi==2021.5.30
cffi==1.14.6
chardet==3.0.4
charset-normalizer==2.0.3
click==7.1.2
cycler==0.10.0
cymem==2.0.5
debugpy==1.3.0
decorator==5.0.9
```

```
defusedxml==0.7.1
Django==3.2.11
docx2txt==0.8
EbookLib==0.17.1
en-core-web-sm @
https://github.com/explosion/spacy-models/releases/download/en_cor
e_web_sm-3.1.0/en_core_web_sm-3.1.0-py3-none-any.whl
entrypoints==0.3
extract-msg==0.23.1
gensim==4.0.1
gitdb==4.0.7
GitPython==3.1.18
idna==3.2
IMAPClient==2.1.0
ipykernel==6.0.3
ipython==7.25.0
ipython-genutils==0.2.0
ipywidgets==7.6.3
jedi==0.18.0
Jinja2==3.0.1
joblib==1.0.1
jsonschema==3.2.0
jupyter-client==6.1.12
jupyter-core==4.7.1
jupyterlab-pygments==0.1.2
jupyterlab-widgets==1.0.0
kiwisolver==1.3.1
```

```
lxml==4.6.5
MarkupSafe==2.0.1
matplotlib==3.4.2
matplotlib-inline==0.1.2
mistune==0.8.4
murmurhash==1.0.5
nbclient==0.5.3
nbconvert==6.1.0
nbformat==5.1.3
nest-asyncio==1.5.1
nltk==3.6.6
notebook==6.4.1
numpy==1.21.1
olefile==0.46
packaging==21.0
pandas==1.2.5
pandocfilters==1.4.3
parso==0.8.2
pathy==0.6.0
pdfminer.six==20181108
pexpect==4.8.0
pickleshare==0.7.5
Pillow==9.0.0
plotly==5.1.0
preshed==3.0.5
prometheus-client==0.11.0
prompt-toolkit==3.0.19
```

```
protobuf==3.17.3
ptyprocess==0.7.0
pyarrow==4.0.1
pycodestyle==2.7.0
pycparser==2.20
pycryptodome==3.10.1
pydantic==1.8.2
pydeck==0.6.2
Pygments==2.9.0
pyparsing==2.4.7
pyrsistent==0.18.0
python-dateutil==2.8.2
python-pptx==0.6.18
pytz==2021.1
pyzmq==22.1.0
regex==2021.7.6
requests==2.26.0
scipy==1.7.0
seaborn==0.11.1
Send2Trash==1.7.1
six==1.12.0
smart-open==5.1.0
smmap==4.0.0
sortedcontainers==2.4.0
soupsieve==2.2.1
spacy==3.1.1
spacy-legacy==3.0.8
```

```
SpeechRecognition==3.8.1
sqlparse==0.4.2
srsly==2.4.1
streamlit==0.84.2
tenacity==8.0.1
terminado==0.10.1
testpath==0.5.0
textdistance==4.2.1
textract==1.6.3
thinc==8.0.8
toml==0.10.2
toolz==0.11.1
tornado==6.1
tqdm==4.61.2
traitlets==5.0.5
typer==0.3.2
typing-extensions==3.10.0.0
tzlocal==1.5.1
urllib3==1.26.6
validators==0.18.2
wasabi==0.8.2
watchdog==2.1.3
wcwidth==0.2.5
webencodings==0.5.1
widgetsnbextension==3.5.1
wordcloud==1.8.1
XlsxWriter==1.4.4
```

7. 5 SIMILAR.PY

```
import textdistance as td


def match(resume, job_des):
   j = td.jaccard.similarity(resume, job_des)
   s = td.sorensen_dice.similarity(resume, job_des)
   c = td.cosine.similarity(resume, job_des)
   o = td.overlap.normalized_similarity(resume, job_des)
   total = (j+s+c+o)/4
   # total = (s+o)/2
   return total*100
```

7.6TFIDF.py

```
from sklearn.feature_extraction.text import
TfidfVectorizer


def do_tfidf(token):
   tfidf = TfidfVectorizer(max_df=0.05, min_df=0.002)
   words = tfidf.fit_transform(token)
   sentence = " ".join(tfidf.get_feature_names())
   return sentence
```

# CHAPTER - 08
# OUTPUT

**Pictures**

# CHAPTER - 09
# REFERENCES

**[1] Nikita Jayakar; Sonia Sunny; Pheba Babu; M. Kiruthika; Ambarish Gurjar Web Application for Screening Resume**

**[2] Taipei, Taiwan ROC ; Han-Yen Yu Based on the application of AI technology in resume analysis and job recommendation**

**[3] B Gunaseelan; Supriya Mandal; V Rajagopalan Automatic Extraction of Segments from Resumes using Machine Learning**

**[4] R Sunitha; K S Varuna; N Varshini; Prasad B Honnavalli Resume Scanning and Emotion Recognition System based on Machine Learning Algorithms**

**[ 5 ] Nimet Tülümen; Gökhan Akgün; Ali Nohutçu; Günnur Sevgi Aktoros Genç; Serdar Genç Hybrid Job and Resume Matcher**