**STEP BY STEP PROCEDURE OF CODE**

**1.1 Working of model**

Initially, the system accepts resume from the job applicant and extracts the text. All types of formats like pdf or file are also accepted. Since many operations cannot be performed from them, they are converted to text first. These are given to the model for training purpose. Then the pre-trained model gets trained on the data that is given as input which is relevant to the system. Once the job applicant submits their resume, it is sent for comparison with the job description requirements. These requirements have already been passed through a pre-trained model, and details such as years of experience and qualifications are already known. The job data is compared with the resume data after which it can calculate the score based on the formula. The score of the resume is calculated based on the comparison with the job description requirements. Finally, this score will help in choosing the correct

**1.2 Training of a Model**

Training of the model includes these following steps

1. Data Collection: Gather a dataset of resumes and corresponding job descriptions from sources like Kaggle.

2. Data Preprocessing: Clean the text data by removing formatting artifacts, personal details, and irrelevant information. Some of the process that include in data preprocessing are

* Cleaning: Remove any unwanted characters, symbols, or formatting artifacts from the text data. This can include HTML tags, punctuation marks, special characters, and numbers.
* Tokenization: Split the text into individual words or tokens. This step divides the text into smaller units, making it easier to process and analyze. Tokens can be words, phrases, or even individual characters, depending on the task.
* Lemmatization: Lemmatization is the process of reducing words to their base or root form. It aims to normalize variations of words to their dictionary form, which can improve the performance of text analysis tasks such as sentiment analysis or topic modeling.
* Stemming: Stemming involves removing suffixes or prefixes from words to reduce them to their root form. Stemming does not always result in actual words but can still be useful for tasks like keyword extraction or search indexing.
* Removal of regular expression: Regular expression removal involves eliminating specific patterns or sequences of characters from text data. This step is useful for cleaning text data by removing unwanted symbols, special characters, or patterns that may not contribute to the analysis.

3. Feature Engineering: Extract relevant features from the text data, such as keywords, skills, and years of experience. This may involve using NLP techniques like named entity recognition and sentiment analysis.

4. Model Selection: Choose a machine learning model suitable for binary classification tasks, such as logistic regression, support vector machines (SVM), or a neural network classifier.

5. Training: Split the dataset into training and validation sets. Train the selected model on the training data, optimizing model parameters using techniques like gradient descent to minimize the loss function.

6. Evaluation: Evaluate the trained model's performance on the validation set using metrics like accuracy, precision, recall, and F1 score. Fine-tune the model based on evaluation results and cross-validation techniques.

**1.3 Testing a Model:**

Testing of the model includes these following steps

1. Data Preparation: Obtain a separate test dataset containing unseen resumes and job descriptions, ensuring it is distinct from the training and validation sets.

2. Data Preprocessing: Preprocess the test data in the same manner as the training data, including cleaning, tokenization, and feature extraction.

3. Model Deployment: Deploy the trained model to a production environment or testing framework where it can process the test data.

4. Inference: Use the deployed model to make predictions on the test dataset. For each resume-job pair, the model predicts the probability of the resume being a good fit for the job based on learned features.

5. Evaluation: Evaluate the model's performance on the test dataset using metrics like accuracy, precision, recall, and AUC. Compare the model's performance to baseline or industry benchmarks to assess its effectiveness in real-world scenarios.

**1.4 Working of program**

Firstly, Import libraries

Import the NumPy library with the name np.

Import the Pandas library with the name pd.

Import the Matplotlib library's pyplot module with the name plt.

Import the Seaborn library with the name sns.

Import the GridSpec class from the Matplotlib library's gridspec module.

The read\_csv() function from the Pandas library is used to read data from a CSV file and load it into a DataFrame. Specify the path to the CSV file as the input parameter to read\_csv(). Then, to display the first 20 rows of the DataFrame, the head() function is used. An integer argument can be specified for displaying the number of rows to the head() function.

Exploratory Data Analysis (EDA) helps in understanding the structure, patterns, and relationships within a dataset. To, print unique categories, identify all unique categories present in the 'Category' column of the dataset. The unique() function is applied to the 'Category' column, which returns an array containing all unique values (categories) in the column. Now print the unique categories in the 'Category' column using data['Category'].unique().

Next, The value\_counts() function is applied to the 'Category' column, which returns a series containing the counts of unique values (categories). Now, print the count of each category using data['Category'].value\_counts().

To visualize the count of each category, Seaborn's countplot() function is used to create a horizontal bar plot showing the count of each category. The 'Category' column is passed as the ‘y’ parameter to create a horizontal bar plot.

Data Visualization is the next process. For that import matplotlib particularly the pyplot module, as it provides the functions necessary for creating visualizations. Next, prepare the data for construction of pie chart. For creating a Subplot Grid, define the layout of the grid using GridSpec from Matplotlib. Then use Matplotlib's pie() function to create a pie chart. This function requires data representing the numeric values to be plotted (category counts) and labels for each category. Then, display the plot plt.show().

Next process is Text cleaning. The clean() function is defined to perform various text cleaning operations on the input text, such as removing URLs, hashtags, mentions, special characters, and non-ASCII characters. For that import the re module, which provides support for regular expressions. Define the clean() function with a single parameter text, representing the input text to be cleaned. Use regular expressions (re.sub()) to perform pattern-based substitution and removal of specific patterns from the text. Each re.sub() call removes a specific pattern from the text. For example:

re.sub('http\S+\s\*', ' ', text): Removes URLs from the text.

re.sub('#\S+', '', text): Removes hashtags (words starting with '#').

re.sub('@\S+', '', text): Removes mentions (words starting with '@').

re.sub('[%s]' % re.escape("""!"#$%&'()\*=,-./:;<=>?@[\]^\_{|}~"""), ' ', text)`: Removes special characters.

re.sub(r'[^\x00-\x7f]', r' ', text): Removes non-ASCII characters.

Next, apply the clean() function to the 'Resume' column. After defining the clean() function, it is applied to each entry in the 'Resume' column of the DataFrame to clean the text data. Use the apply() method of the DataFrame to apply the clean() function to each entry in the 'Resume' column. Store the cleaned text in a new column named 'clean text'.

Word Cloud is an important process

Tokenization breaks down text into smaller units, such as words or sentences. In this step, tokenize the cleaned text data to prepare it for further processing. For that import word\_tokenize from the NLTK library. Then, tokenize the cleaned text data using NLTK's word\_tokenize() function.

Then, compute the frequency distribution of words and extract the 100 most common words. Generate a Word Cloud from the cleaned text using the WordCloud library. For that import FreqDist from the NLTK library. Then, flatten the list of tokens into a single list. Use the FreqDist() function to compute the frequency distribution of words. Extract the 100 most common words using the most\_common() method. Then, generate a Word Cloud from the cleaned text. For that import WordCloud from the WordCloud library. Initialize a WordCloud object with desired parameters, such as width, height, background\_color, etc. Use the generate() method to generate the Word Cloud from the cleaned text.

Use LabelEncoder from scikit-learn to convert categorical labels ('Category') into numerical values. It converts categorical labels into numerical values, which can be fed into machine learning algorithms.

Initialize a TF-IDF vectorizer (TfidfVectorizer) with specified parameters. Text Vectorization converts text data into numerical features that can be used as input for machine learning algorithms. TF-IDF (Term Frequency-Inverse Document Frequency) is a popular technique for text vectorization. Fit the vectorizer on the cleaned text data and transform it into a sparse matrix of TF-IDF features.

Train-Test Split is imported from scikit-learn. Train-Test Split is used to divide the dataset into training and testing sets to assess the performance of the machine learning model. Split the TF-IDF features and target labels into training and testing sets using train\_test\_split().

Next for model training and evaluation, import OneVsRestClassifier and KNeighborsClassifier from scikit-learn. Initialize an instance of KNeighborsClassifier. Initialize an instance of OneVsRestClassifier, passing the KNeighborsClassifier as the estimator parameter.

For training model on train data, use the fit() method of the model object, passing the training features (x\_train) and target labels (y\_train).

For predicting the target labels for text data use the predict() method of the model object, passing the test features (x\_test).

For calculating and printing the training and testing scores, use the score() method of the model object to calculate the accuracy score on both the training and testing data.

Print the calculated scores.

Finally, print a classification report showing precision, recall, F1-score, and support for each class. For that Import classification\_report from scikit-learn. Then, use the classification\_report() function, passing the true labels (y\_test) and predicted labels (prediction).