Introduction

**Current Project**:

Fractionable tool has ability to extract resumes from 5 sources i.e., Dice, Monster Jobs, Naukri. etc. the extracted resumes so far are approximately above 1300 resumes, since we have a huge database, we want to use the data and build an AI which takes the Job description given by recruiter and searches through the data warehouse for all the suitable resumes or profiles and gives them a score based on the JD.

**Model:**

Version 1: NER on whole resume

* When the resumes are trained and tested using NER model the accuracy generated is around 68%.
* Model consisted of 21 entities.

Firstly, NER model was built but it has few challenges:

* Not performing well when resume consists of tables.
* Did not perform well on when the resume template consists of content in columns.

The possible solution for the problems from the version1 was to split the resume based on their section heading or certain entities and save them into separate buckets based on paragraph headings and then training separate models on those: namely Personal details, Experience, Education, Overview with each model having accuracy of 89%, 73%, 68% respectively as of using the extraction method of using “pymupdf” and separating the sections and paragraphs.

Version 2: Split method

The code for separating the entities was written dynamically, but we cannot automate it, to overcome this problem, we are suggesting use of a new model i.e., Layout Parser.

Layout Parser has been trained on more than 2 lakhs resumes of various formats, to incorporate our problem and make use of layout parser in custom format, we must annotate the resumes based on our requirements using Label Studio. At least 300-400 resumes or documents are recommended but our goal is to at least annotate 600 resumes for better performance and incorporate diverse document types.

The model takes images as input therefore we need to convert the pdfs into images.

Below I am presenting my tasks and how I executed them:

**Task 1 - Annotations**

* 1. Task Definition

Annotations of resumes is required for training a Layout Parser model which is pre-trained, to achieve our desired output, it is required to train the model on our custom data and format. To fulfil that, the annotations task was carried out. Below are pre-requisites and steps taken to perform successful annotations.

To annotate we must install **Labelstudio**.

Steps:

1. Open anaconda prompt and run following commands:

*Conda create –n label-studio python=3.9 #to create a separate environment*

*Conda activate label-studio #activate the environment.*

*Pip install label-studio #installing label studio*

1. Launch the Label-studio from anaconda prompt using following command:

*Label-studio. #launching the label-studio*

1. Sign up > Create a project > Set up
2. Go to Setting > Label Interface> in the Code section:

*<View>  
<Image name="image" value="$image"/>  
<RectangleLabels name="label" toName="image">  
  
  
<Label value="Header\_1" background="#FFA39E"/><Label value="Header\_2" background="#D4380D"/><Label value="Text" background="#FFC069"/><Label value="Table" background="#AD8B00"/><Label value="Image" background="#FFA39E"/></RectangleLabels>  
</View>*

**Paste the above code**.

1. Then go to Visual Section:

Change the names of the tags: Section Heading, Paragraph, Table, Images.

Save the settings.

To Annotate:

**Import data**: first convert all the resumes from pdf format to images, then import them into ur project and start annotating.

**Note:** The annotations must be very precise since we are dealing with text, annotating an empty space would be considered by model as \n or \t or \s

**Task 2: Data Extraction**

**2.1 Task Definition**

Data Collection for identifying the section headers generated from the layout parser model is an important task to get the relevant data. The data is extracted from the provided resumes by going through certain conditions and constraints. The data consists of all possible Section headings and their relevant tags/Labels which is used by Word Phrase Classifier model which is trained to identify the section headings and categorize it into a specific bucket, from a resume. This is required to extract the subsequent text or paragraph content beneath the Section headings or Sub-headings.

**2.2 Algorithm Definition**

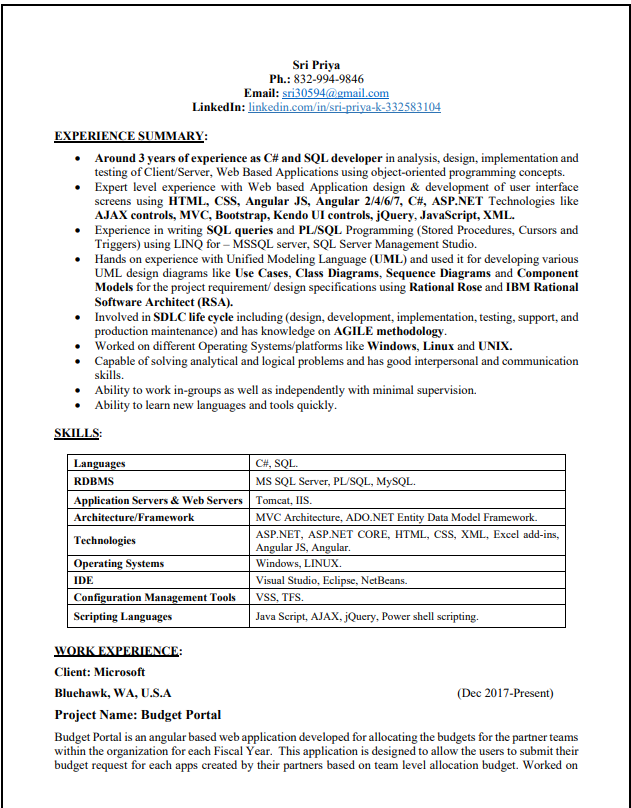
Method 1: Dynamic Python Code

Data Collection for Word Phrase Classification is the problem addressed. The input for the Algorithm is the raw resumes running through loop and running each line of the resume to identify the section heading, by going through few constraints. The output for this is the extracted possible Section headings from each resume.

A conditional based algorithm is used with custom constraints to extract the Section headings from each resume. The resumes are split into lines and each line runs through a set of constraints as mentioned below:

* The line must begin with a Capitalized or Upper-Case word.
* The line must not contain any stop words except “The”.
* We are replacing the possible punctuations used in a Section Header i.e., [“: ” , “ ,”, “ .”, “-“] with blank or nothing.
* The line must contain words no more than 4.
* The line must contain only Alphabets and no numeric content.

For example:



This the first page of a resume, the resume goes through the constraints mentioned above and extracts the following phrases:

Text

Description automatically generated

Among these the most relevant Data which we are going to use according to our problem statement are the “EXPERIENCE SUMMARY”, “SKILLS” and “WORK EXPERIENCE”. The rest of the data is irrelevant (only considering section headings from the example above).

The dependent variable are the tags which are given manually to each section heading extracted, as of now (at the time of extraction) there are 16 labels or dependent variable categories. The independent variable is the Section headings that are extracted.

The relevant extracted data is very less compared to extraction of noise but that cannot be avoided since the each resumes contain only approximately 6-7 section headings only, whereas the paragraph , lines, side content, tables are all considered by the algorithm as section heading as well, because each resume may contain several lines coinciding with the mentioned or considered constraints used for the data extracted by the custom algorithm.

Refer code from file: Fractionable AI\Pre-processing\method 1\_data extraction\_\Data Extraction from Resumes\_method1.ipynb

Method 2: Using PymuPDF module

Since the method one did not yield proper data required for model training, we went with a second approach using the pymupdf and Fitz modules, (Note: Install fitz before installing PymuPDF package).

Open the documents using fitz module, and extract all the text from pdfs, based on the assumption that headers and paragraphs are often separated by the **font size** and **font weight** of the text and that the **most used font** can be considered the paragraph.

* Using PymuPDF we identify the paragraphs as text with the most used font in the document, headers as anything larger, and subscripts as those smaller than the paragraph style.
* Then we create a dictionary with HTML style element tags such as <h1>, <p> and <s0> for the headers, paragraphs, and subscripts respectively.
* Finally, Annotate pieces of text with these element <tags>.

The input for this method is resumes in pdf format and the output the html tagged text/ section headings.

**Conclusion:**

This method fails because of the assumption that different elements of the text are divided by the font size and shape, but it was found that nearly 80% of resumes provided did not satisfy the assumptions and so resulted in poor data.

Refer code from file: Fractionable AI\Pre-processing\method 2\_data extraction\_pymupdf\Section\_heading extractions using pymuPDF\_method2.ipynb

Method 3: Using pytesseract to extract section headings from the bounding boxes.

Since the above 2 methods failed, we tried to extract the section headings directly from the annotated data of our resumes by using the json file as input and extracting the bounding boxes of the section heading, we then incorporated a logic to fit the bounding box coordinates onto the corresponding image of the resumes, followed by extracting the text from within the bounding box explicitly by using “pytesseract”.

After extracting the section headings, the text was pre-processed and saved in excel file.

Challenge here was wrongly annotated data, was also extracted as section heading and had to be corrected by the annotator. After extraction the data was found to contain too many duplicates and the quantity was less.

**Conclusion:**

It is better to use the data extracted from bounding boxes of our annotated data and add any missing elements explicitly and train the model for better performance. However, it led to extraction of wrong section headings which when trained will result in misleading results.

Refer code from file:

Fractionable AI\Pre-processing\method 3\_data extraction\_bbox\Extract text from Bounding box\_method3.ipynb

Method 4: Creating synthetic data.

The above mentioned Methods were not yielding desired output and were very time consuming in terms of first of extracting the section headings and later labelling them, the only solution we could find was to create a list of all possible section headings(note: it is non exhaustive, future additions can be made based on data requirements), the sections heading were created a dictionary and saved in excel format later, to be used in training.

Conclusion:

This method of Data Creation was reliable and yielded better model performance.

Refer code from file: Creation of Standard Data.ipynb

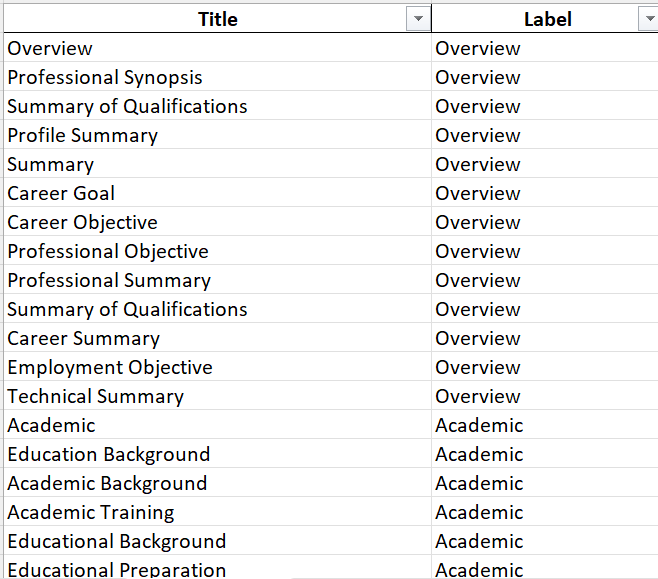
**Task 3: Building Word Phrase Classification Model**

**Introduction:**

After Data collection of possible “section headings” of a resume, the data extracted acts as an input into the Word Phrase Classifier Model.

Word Phrase Classifier Model is a custom model to identify the section headings and label them their respective category.

For example,



Given the Title or Section Heading from a resume, the model should be able to identify its possible category, in our case we initially decided with only 7 Categories which are: Overview, Academic, Work Experience, Activities, Skills, Certifications, Achievements.

We experimented with various models and chose the model with better performance; I am going to include details on each model and significant modifications made to achieve better performance.

**Model 1: NaïveBayes Classifier:**

**3.1.1 Algorithm Definition:**

**Naive Bayes** classifiers are a collection of classification algorithms based on **Bayes’ Theorem**. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e., every pair of features being classified is independent of each other.

**Naive Bayes** classifiers have been heavily used for **text classification** and **text** **analysis** machine learning **problems**. Since, it follows the principle of prediction based on the probabilities, it best suits for text classification. Hence, we performed Naïve Bayes Classification for our Word Phrase Classification Model.

The model was run 2 separate data to test the performance 1. Data extracted from bounding boxes 2. Data extracted using Pymupdf.

**3.1.2 Experimental Evaluation:**

**Methodology:**

The text data was converted into vectors using TF-idf vectorizer and trained the vectorised data using Naïve Bayes Classifier.

**Results:**

The model could not capture the pattern in the data and failed to predict the right category of the section\_headings, resulting on only 27% accuracy.

A screenshot of a computer

Description automatically generated with low confidence

**Model 2**: **Word2Vec by gensim**

**3.2.1 Algorithm Definition:**

To build Word Phrase Classification model, we initially decided to build the model using gensim Word2Vec model. Word2Vec algorithm is a natural language processing technique by Google. It consists of models used for mapping words to vectors of real numbers, or in other words, for generating embeddings. The basic idea behind word embeddings is that words that occur in a similar context in a text tend to be closer to each other in vector space. This concept is exactly what we need to incorporate for our Word Phrase Classifier model, since similar section\_headings would have same context therefore it would be easy for model to identify the Category if the section\_headings from our resume.

**3.2.2 Experimental Evaluation**

**Methodology**

Word2vec model is nicely implemented in the [*Gensim*library](https://radimrehurek.com/gensim/models/word2vec.html). The data for the model should be in the list of lists form where every word from a section heading is an item of a list. So, the corpus is built from our data in a way that each section heading is list of lists consisting of individual words, for example: “Professional summary” in the corpus should be in the following format i.e., [[“Professional”,” summary”]].

Then, we compile and train a word2vec model. The model produces high-dimensional vectors, where the s*ize*parameter sets the number of dimensions. The optimal number of dimensions depends on the size of the dataset. In our case, we have set 100 dimensions. *min\_count*parameter is set to 1, which controls the minimum frequency of words. Model training ignores all words with a total frequency lower than this value.

After the model is built, we perform PCA for dimensionality reduction, since our trained model has 100 dimensions, to further cluster our data based on semantic meaning. After reducing the dimensions, K means clustering is performed and each cluster consists of similar semantic meaning.

**Results:**

The Word2vec model did not perform well in this experiment possible reason could be less amount of data since our corpus consisted of only 87 unique words, which is insufficient to train a high-level model like word2vec. The yielded results did not work as expected.

A picture containing text, screenshot, font, number

Description automatically generated

Below is the graphical representation of obtained clusters of our corpus.

A picture containing screenshot, colorfulness, diagram, plot

Description automatically generated

The above cluster has no relationship among themselves, these maybe relevant in terms of normal English corpus but in context with the section headings the clusters had no semantic meaning among them.

**Model 3: fasttext model**

**3.3.1 Algorithm Definition:**

fastText is an open-source library, developed by the Facebook AI Research lab. Fasttext module is known for its accurate result in quick time. It achieves this computational efficiency and accuracy by employing 2 methods to address classification and training word representations of text which are Hierarchical softmax and word n-gram. In order to train and evaluate this classifier, we’ll have to prepare our data in a format fastText expects.

fastText expects the category first, with the prefix ‘\_\_label\_\_’ before each category, and then the input text, like so,

For example, A picture containing text, screenshot, font, receipt

Description automatically generated

The model expects the data to be in format like label\_description which is the training data format.

**3.3.2 Experimental Evaluation:**

**Methodology:**

The data used for training fasttext model is Standard Data, the data is further cleaned and transformed to fit the prerequisites of the fasttext model. After the transformation, data is pre-processed by removing any possible punctuations and stopwords for faster and accurate training followed by splitting into train and test data. To save the train and test data, it is required to convert train and test data to separate csv and save with the extensions “. train” or “.test”. The train data is then trained on supervised fasttext model.

The training was performed on imbalanced data, the class with wring predictions were given more priority in terms of training data count for the model to learn well and learn the unlearned class.

**Results:**

Fasttext model outperformed other text classification methods and model for our data and achieved 90% accuracy on train data.

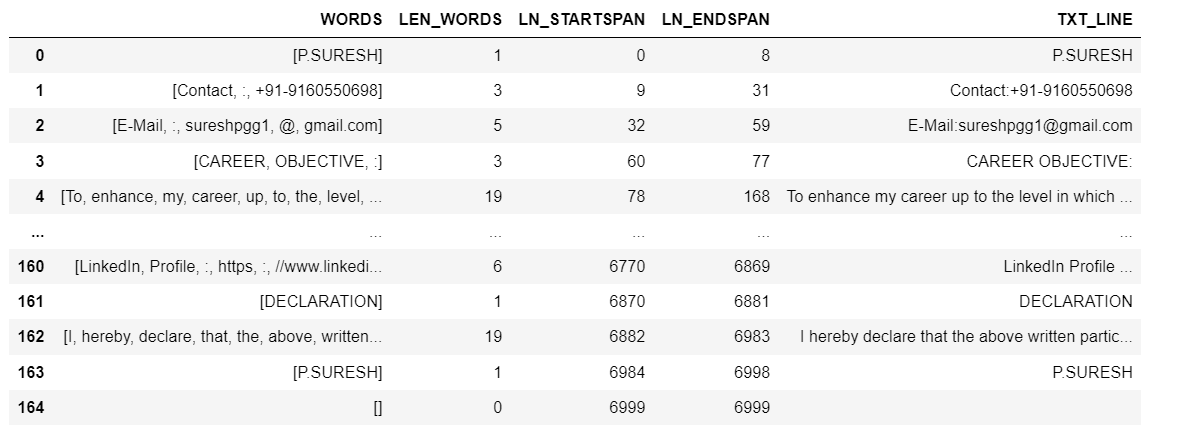
A screenshot of a computer

Description automatically generated with low confidence

**Task 5: Extracting texts from section headings**

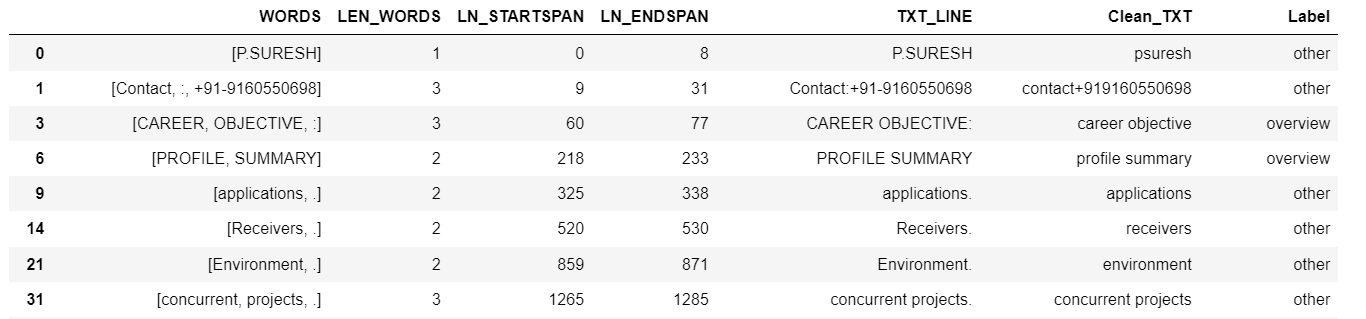
**Methodology:**

After successfully training a model to identify the sections headings from the resume, the next step in our pipeline is to extract the data beneath the section headings and save them in their respective buckets. This step is necessary to train individual spacy models on the split data. The input for this is the resume in pdf format. We first read the resume (pdf format) using fitz module. After reading the resume, text is extracted from the document. The extracted text is split using lines, each line is then detokenized into words. During this process we also extract the data like words(detokized line), length of words in a line, the start index of the line and end index and the line itself. This data is the transformed into a Dataframe. For example,



After forming into dataframe, we filter all the lines whose length is less than or equal to fur, because conventionally section headings do not include more than 4 words. After filtering the data undergoes data cleaning like removing punctuations, converting the text to lower case, and including only alphanumeric characters.

Then we load the model on which the data was trained to identify section headings. After loading we run the predictions on the clean text and the result is added as label to the data frame.



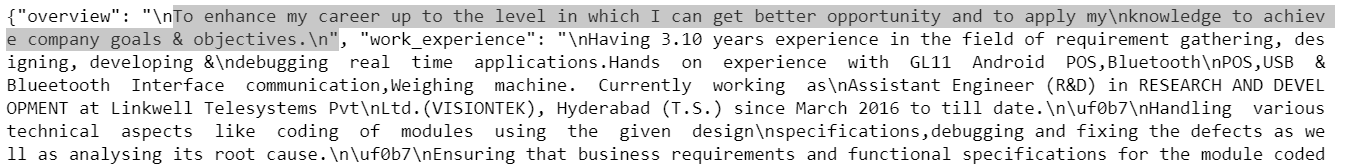
Then any line that is labelled as other is dropped, because it is not relevant to us. Then we extract the data from the end span of the first section heading to the start span of the next corresponding section headings. This is the process of splitting the resumes based on their category and saving them in different buckets for further training individual spacy models.

For example,

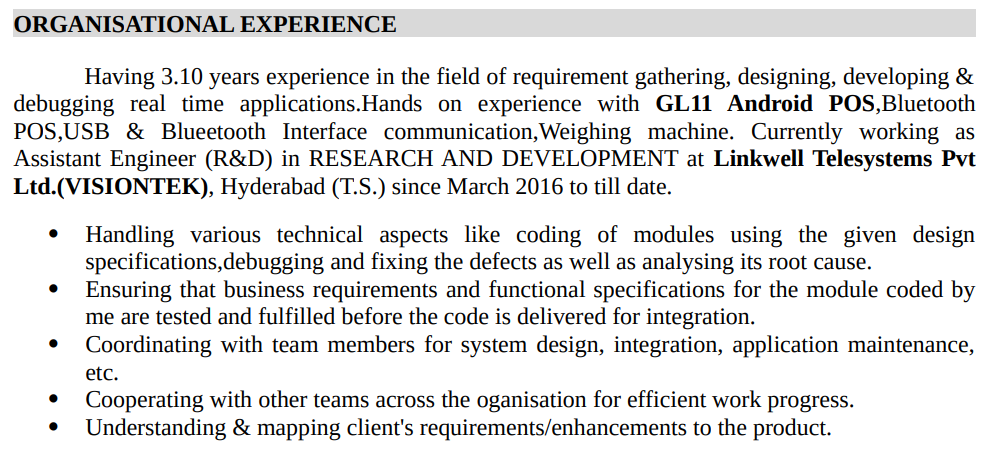
* The text under Career Objective is saved to overview bucket.

A picture containing text, screenshot, font, line

Description automatically generated



* The organisational experience which is work experience is saved to work experience bucket.



A close-up of a text

Description automatically generated with low confidence

Task 6: SJD Sourcing strategy

Introduction:

The Sourcing strategy is the next step after building a nlp model to extract the features and train them from our resumes. The goal of sourcing strategy is to be able to suggest what kind of candidates to target given an SJD and what measures to take to ensure that the candidate is the right choice for the given SJD role. The sourcing strategy data comes from Fractionable tool, where the recruiters mention the requirements and what kind of evaluation criteria the candidate must satisfy, the number of years of experience a candidate must have for that role and previous employer, etc.

**Model 1: Trained old Fractionable data using Pycaret**

Methodology:

The data for sourcing strategy is gathered from the Fractionable tool database, the data consists of 1264 rows and 22 columns, of which our main targets or dependent variables are Source(from which portal to source the resumes for the given SJD), evaluationCriteria(what are the steps to be taken , like how many rounds of interview, written test, coding/hackerrank test, aptitude test, etc), Candidate Experience(years of experience, Target Company(What kind of companies to target for given SJD). The given data then was trained on mainly 3 models using the Pycaret (AutoML tool).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | Target | Best Model | Results | Comments |
| Model 1: Classification Model | Source | Random forest classifier | Accuracy: 70% | Istechnicallyselected = True |
| Model 2: Classification | evaluationCriteria | Extra Tress Classifier | Accuracy: 97% | Istechnicallyselected = True, The data is very irrelevant and dirty. |
| Model 3: Regression | Candidate experience | Random forest regressor | R2: 36% | Istechnicallyselected = True, The data is very discrete, so the model could not learn the regression trends. |

Model 1:

A picture containing text, screenshot, font, number

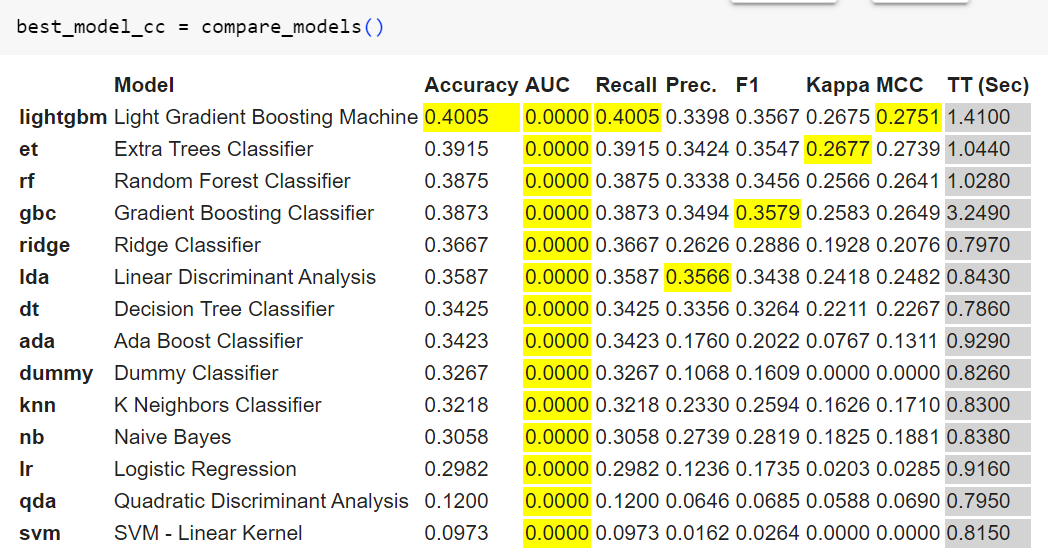
Description automatically generated

Model 2:

A screenshot of a computer

Description automatically generated with medium confidence

Model 3:



Since the models did not perform well, we performed clustering on the data to retrieve any emerging patterns for which the results are attached below. There were no emerging patterns found with the given data.

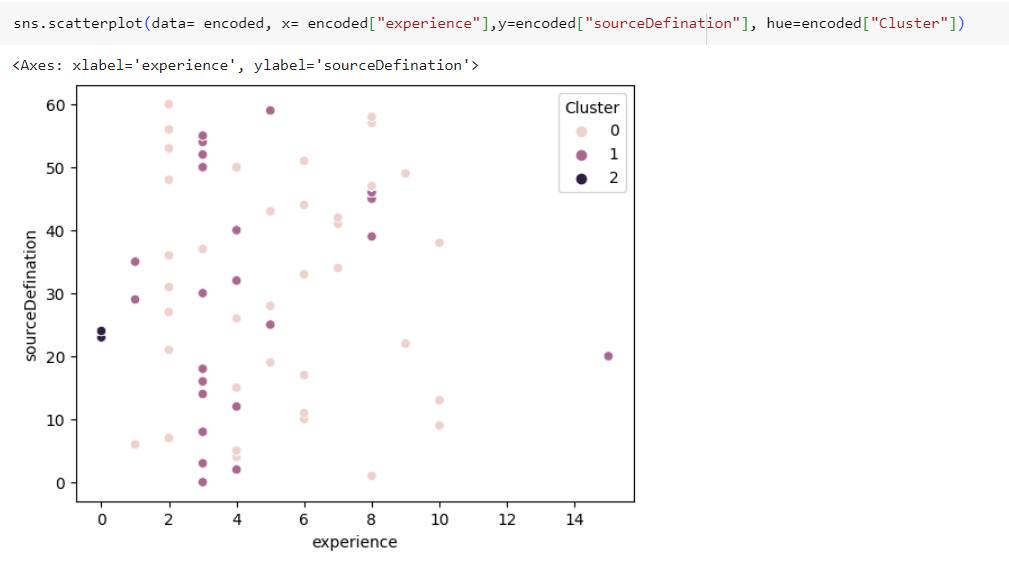
HeatMap showing any relationship between the data:

A picture containing screenshot, text, square, pattern

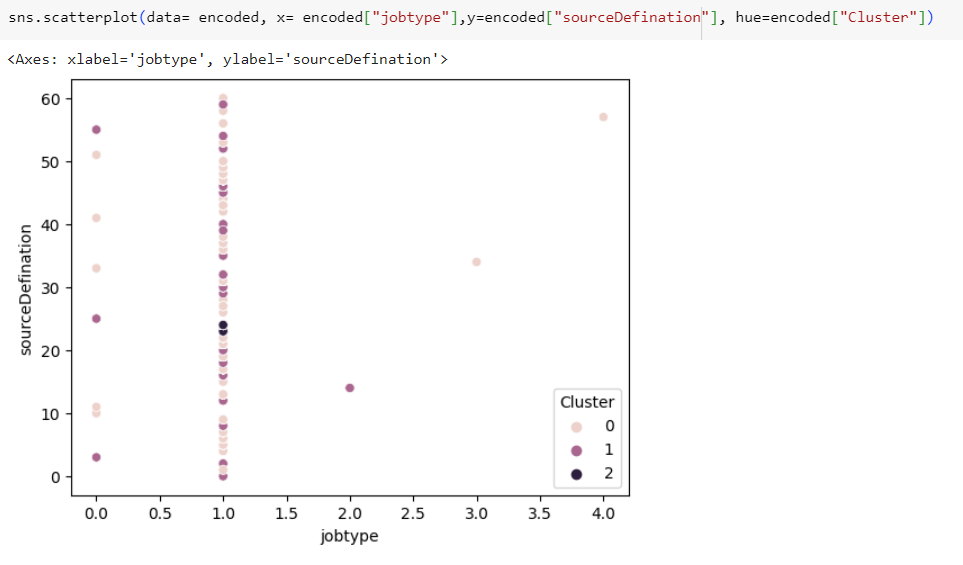
Description automatically generated

Below are the results of K-mode clustering performed on the data:

**Experience vs SourceDefinition( SJD)**



**Jobtype vs sourceDefinition(SJD)**



Operationstype vs sourceDefinition(SJD)

A screenshot of a computer screen

Description automatically generated with medium confidence

**Experience vs sourceDefinition(sjd)**

A screen shot of a graph

Description automatically generated with medium confidence

Conclusion:

As seen above the data is inconsistent and insufficient for us to train a good model and extract any emerging patterns.

Model 2: New SJD data with Clustering

Methodology:

The second model was trained on new SJD data since the previous data was insufficient, which consists of 1876 rows and 39 columns, out of which we have selected on few columns which are relevant to our problem statement. Since the data was found to be inconsistent (in terms of lot of duplicate which needs in depth analysis), after the cleaning for the model training, we have few features for sourcing strategy which are to be given by the model given the SJD: sourcingChannel, experienceLevel, keywords, targetCompanies, evaluationCriteria, location, but there is no relevant data for all the columns so the columns to be predicted by model are:

The final **dependent variables** are:

- Portal - Candidate

- Experience required(range)

- Target Companies- Candidate

- EvaluationCriteria – Candidate

And the independent variables are:

**Independent Variables:**

- Source Definition Name - SJD

- Skills - Candidate

- Highest Education - Candidate

- Experience - Candidate

- Location - SJD

- Client - SJD

- Jobtype - SJD

- Paytype – SJD

Results:

After performing clustering on the given data below are the results found.

A picture containing text, screenshot, diagram, line

Description automatically generated

A picture containing text, screenshot, rectangle, number

Description automatically generated

A picture containing text, screenshot, diagram, parallel

Description automatically generated

A picture containing text, screenshot, diagram, plot

Description automatically generated