A Project Work - Phase II on

Automated Word Formatter

Submitted in partial fulfillment of the requirements for the award of the

Bachelor of Technology

in

Department of Computer Science and Engineering
(Artificial Intelligence and Machine Learning)

by

N. Krishna Kalyan	**20241A6638**
Abhinav Aka	**20241A6605**
Raya Samuel Sajit	**20241A6647**

Under the Esteemed guidance of

Dr. M. KiranKumar

Associate Professor

Department of Computer Science and Engineering

(Artificial Intelligence and Machine Learning)

GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY

(Approved by AICTE, Autonomous under JNTUH, Hyderabad)

Bachupally, Kukatpally, Hyderabad-500090

GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY
(Autonomous)

Hyderabad-500090

CERTIFICATE

This is to certify that the Project Work - Phase II entitled "Automated Word Formatter" is submitted by **N. Krishna Kalyan (20241A6638), Aka Abhinav (20241A6605) and Raya Samuel Sajit (20241A6647)** in partial fulfillment of the award of degree in BACHELOR OF TECHNOLOGY in Computer Science and Engineering (Artificial Intelligence and Machine Learning) during Academic year 2023-2024.

Internal Guide	Head of the Department
Dr. M. Kiran Kumar	Dr. G. Karuna

External Examiner

ACKNOWLEDGEMENT

There are many people who helped us directly and indirectly to complete our Project Work - Phase II successfully. We would like to take this opportunity to thank one and all. First, we would like to express our deep gratitude towards our internal guide **Dr. M. Kiran Kumar, Associate Professor,** Department of Computer Science and Engineering (Artificial Intelligence and Machine Learning), for his support in the completion of our dissertation. We wish to express our sincere thanks to **Dr. G. Karuna,** Head of the Department**,** and to our principal **Dr. J. PRAVEEN,** for providing the facilities to complete the dissertation. We would like to thank Project Coordinator, **Dr. R. P. Ram Kumar**, our faculty members and friends for their help and constructive criticism during the period. Finally, we are very much indebted to our parents for their moral support and encouragement to achieve goals.

N. Krishna Kalyan (20241A6638)
Aka Abhinav (20241A6605)
Raya Samuel Sajit (20241A6647)

DECLARATION

We hereby declare that the Project Work - Phase II titled **"Automated Word Formatter"** is the work done during the period from **17th July 2023 to 23th November 2023** and is submitted in the partial fulfillment of the requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering (Artificial Intelligence and Machine Learning) from Gokaraju Rangaraju Institute of Engineering and Technology (Autonomous under Jawaharlal Nehru Technology University, Hyderabad). The results embodied in this Project Work - Phase II have not been submitted to any other University or Institution for the award of any degree or diploma.

N. Krishna Kalyan (20241A6638)
Aka Abhinav (20241A6605)
Raya Samuel Sajit (20241A6647)

ABSTRACT

Researches face a lot of problem while documenting their Manuscripts. In absence of proper formatting, the research paper may get rejected, even if it has quality content. The aim of project is to create an application to streamline the process of transforming documents into various recognized academic or professional styles, ensuring consistency and precision. The proposed application is useful for a wide range of usecases, from Research papers to Academic Thesis, which effortlessly manages titles, headings, text, tables, figures, equations and also presents the content in an organized and professional manner. Additionally, it automates the handling of Citations and References, significantly enhancing the document preparation efficiently across various formatting standards. The proposed application is constructed by modifying the nodes of the document tree structure. With this versatile application, valuable time and effort can be saved while ensuring the documents adhere to the highest standards of formatting.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

Acronym	Full Form
E3S	Electronic Entertainment Experience
IEEE	Institute of Electrical and Electronics Engineers
MVT	Model-View-Template
XG Boost	extreme Gradient Boosting
Mongo DB	Mongo Data Base
Latex	Lamport's Tex
.docx	document extended
SecTag	Section Tagging

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Automated Word Formatter

In today's academic and professional world, the correct formatting of documents according to research paper standards is crucial for success. Whether you're a student, researcher, or professional, your work often demands adherence to specific guidelines, such as those set by renowned organizations like the Electronic Entertainment Experience (E3S). Achieving impeccable formatting, however, can be a daunting and time-consuming task, leading to frustration and potential errors. To address this challenge, we present the Automated Word Formatter, a web-based platform designed to simplify the process of correctly formatting your documents. At its core, the Automated Word Formatter streamlines the way you format word documents according to research paper standards, such as those outlined by the E3S. By providing users with a seamless and efficient solution, this platform empowers individuals to focus on the substance of their work rather than getting lost in the intricacies of formatting. The Automated Word Formatter offers an intuitive and user-friendly interface, making it accessible to a wide range of users, from students and researchers to professionals across various fields. The key objective is to eliminate the hassle and complexities associated with formatting, ultimately saving time and reducing the risk of errors.

This platform operates on a simple premise: a user uploads a word document, and the Automated Word Formatter takes care of the rest. By leveraging advanced algorithms and pre-defined formatting styles, the system automatically refines the document's layout, ensuring it adheres to the exacting standards expected in the world of research papers. Whether it's managing fonts, margins, headings, or reference lists, the Automated Word Formatter excels in producing professional and standardized documents. The automated formatting tool caters to a variety of requirements and guidelines, with a primary focus on the stringent standards set forth by the E3S. This ensures that your work aligns with the expectations of prestigious institutions and publications. We understand that conformity to these standards is essential not only for readability but also for fostering credibility and trust in your research. Furthermore, the Automated Word Formatter is underpinned by a robust back-end system that securely processes your documents. Your data is treated with the utmost care and confidentiality, allowing you to trust our platform for your sensitive research work. In a nutshell, the Automated Word

Formatter is your ultimate partner in document formatting, simplifying what was once a cumbersome and time-intensive task. It allows you to maintain your focus on the core of your work, all while ensuring that your research paper adheres to the highest standards. Whether you're new to academic writing or a seasoned professional, our platform welcomes you to experience the future of document formatting. Say goodbye to the stress of document formatting; say hello to the Automated Word Formatter, your gateway to impeccable and effortless research paper formatting.

1.2 Objective of the Project

To Implement a python-based word formatting tool which automates manual editing to save time and effort.

The approach used is to classify the elements of the document this process is carried out by analyzing the xml structure of the document.

To implement approaches for formatting text, images, and tables and configuring the type of text and its style by using a XG Boost classifier model.

1.3 Methodology

The objectives are achieved by performing the following steps

Step 1: Saving the Document in the MongoDB Cloud Database

Figure 1.1 Document Upload

In this initial step as shown in fig 1.1, is to upload the document to the MongoDB database after which the document undergoes ingestion and is securely stored within the application's MongoDB cloud database. Renowned for its robust web framework, the application seamlessly integrates with MongoDB, a sophisticated cloud-based database system. This strategic storage choice is particularly beneficial when handling unstructured data

types, such as documents. By employing this MongoDB cloud database, the application ensures seamless accessibility and efficient processing of the document, setting the stage for subsequent steps in the formatting.

Step 2: Data Extraction Using the Python Docx Module

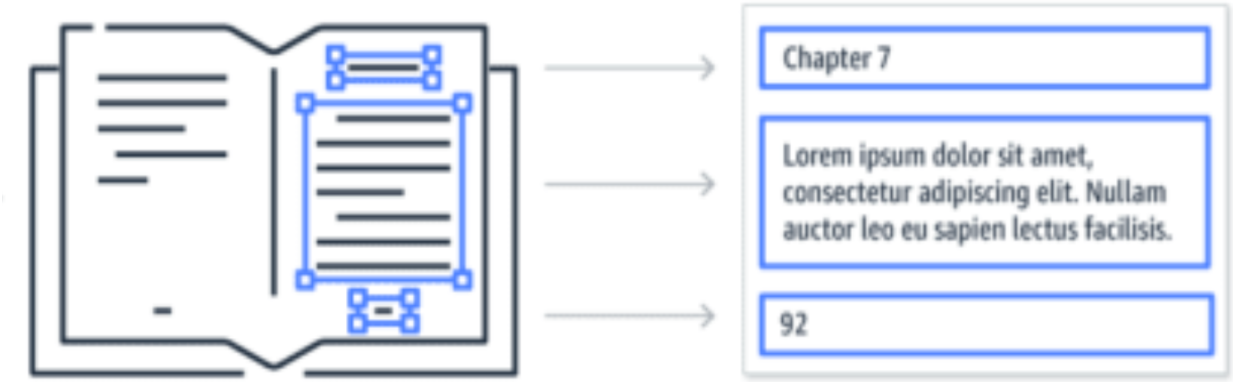

Figure 1.2 Data Extraction

Once securely storing the document in the database, the subsequent phase, illustrated in Figure 1.2, centres around the pivotal process of data extraction. This crucial step is meticulously orchestrated through the adept use of the Python Docx module—an exceptionally versatile library crafted for the precise parsing of Word documents. Revered for its robust functionality, this module takes on the intricate task of unraveling both the textual nuances and structural intricacies woven into the document's fabric. Navigating through the document's complexities, the Python Docx module seamlessly retrieves essential content, laying the groundwork for subsequent stages of sophisticated processing and analysis. This refined methodology not only safeguards the document's integrity but also establishes the framework for deploying advanced techniques and algorithms in the ensuing phases of the document formatting pipeline.

Step 3: Classification of Elements as Headings, Non-Headings, Tables or Images.

Figure 1.3 Classification of Elements

As demonstrated in fig 1.3 once the document's content extracted, the system employs the XGBoost classifier, a machine learning algorithm, to categorize elements within the document. The classifier differentiates between headings (such as titles, section headings, or subsections), non-headings (regular text, paragraphs, etc.), Tables and Images. This classification process has the following involved

1. Identification of Special Elements:

The XGBoost classifier is trained to recognize various special elements within the document, including citations, footnotes, and references. This capability ensures that specific formatting rules can be applied accurately to these distinct content types, maintaining consistency and clarity in the final formatted document.

2. Handling of Text Styles:

In addition to categorization, the classifier can detect different text styles such as bold, italicized, or underlined text. This information is invaluable for applying consistent formatting styles across the document, ensuring a polished and professional appearance.

3. Structural Analysis for Layout Adjustment:

Beyond element categorization, the system performs structural analysis to understand the document's hierarchy. This involves identifying nested sections, lists, and hierarchical relationships between headings and subheadings. Such insights enable precise layout adjustments based on the document's structural organization, ensuring optimal presentation.

4. Optimized Formatting Workflow:

The automation of document element categorization streamlines the workflow for applying formatting styles. By reducing manual effort and enabling scalable processing, the system facilitates efficient adherence to formatting standards across a large volume of documents, ultimately enhancing productivity and consistency in document preparation.

Step 4: Applying Predefined Styles Using the Python Regex Module

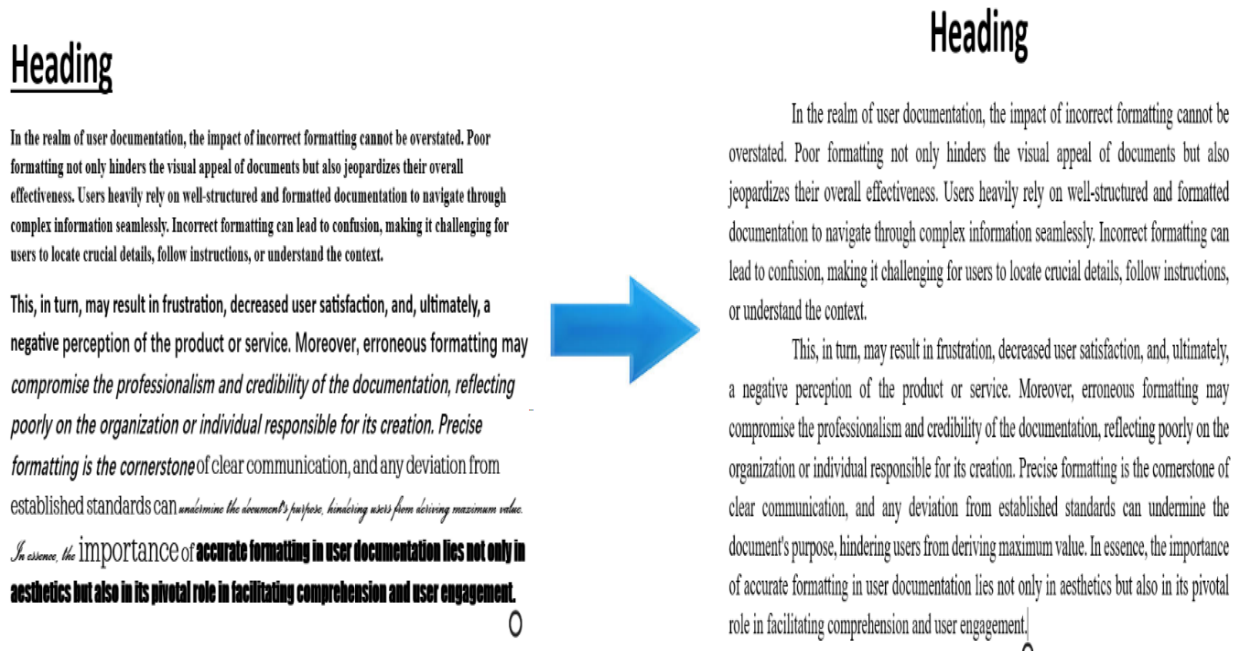

Figure 1.4 style application

In Figure 1.4, following the successful categorization of document elements, the system seamlessly incorporates the Python regex module into the workflow. This module, leveraging regular expressions (regex) within Python, plays a pivotal role in identifying and locating specific patterns embedded within the document content. These discerned patterns

correspond to predefined formatting styles crucial for various elements identified during the classification process. The strategic use of regex empowers the system to efficiently navigate through the document, accurately identifying formatting cues essential for adhering to the rigorous standards set forth in research paper guidelines. The flexible and precise nature of regex enhances the system's ability to apply nuanced formatting styles to diverse elements, contributing significantly to the document's overall coherence and professionalism.

Step 5: Adjusting Page Layout Using the Docx Module

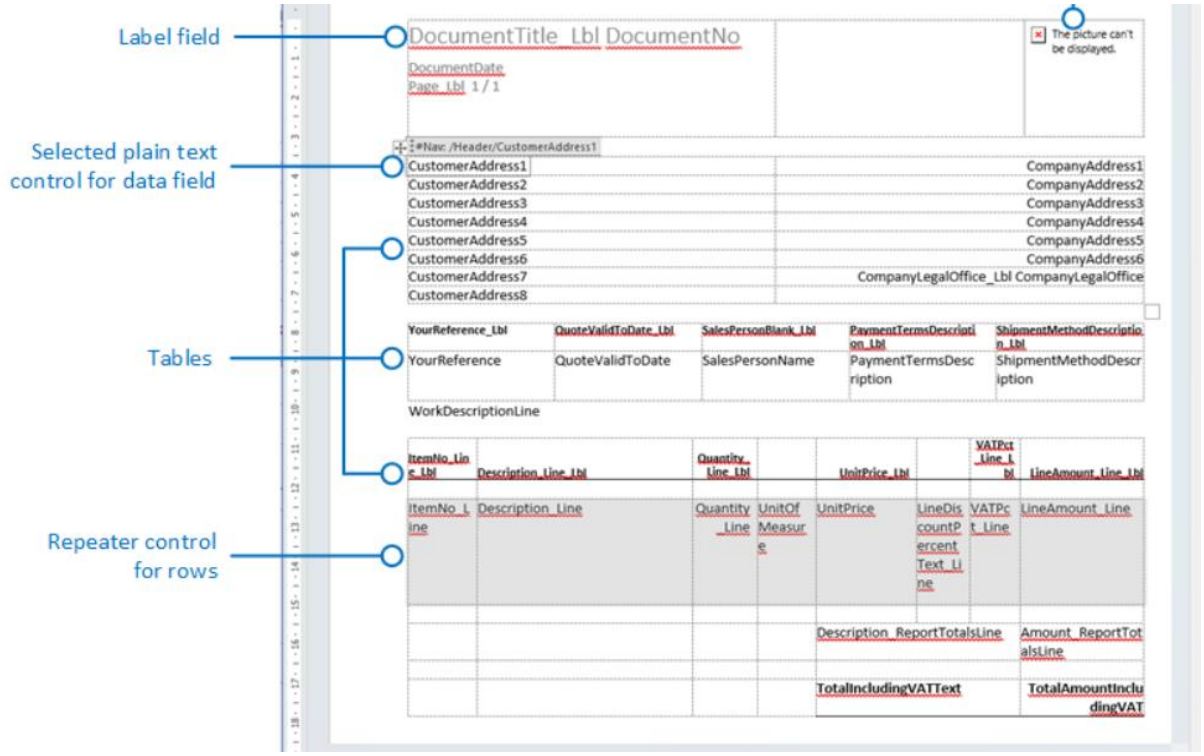

Figure 1.5 Layout Adjustment

As illustrated in fig1.5, the final phase of the methodology involves using the Python Docx module is once again this time for page layout adjustments. Elements such as

1. Margins and Spacing:

Adjusting margins is essential for controlling the amount of white space around the content, which affects readability and aesthetics. Proper margins (top, bottom, left, and right) are set to accommodate the specified formatting guidelines.

2. Indentation:

Indentation is used to define the structure of the document, particularly in paragraphs, quotes, and lists. Consistent and appropriate indentation levels enhance clarity and organization.

3. Alignment:

Text alignment (left, right, center, or justified) ensures a uniform appearance and readability throughout the document. Each section's alignment can be adjusted to match the chosen formatting style.

4. Line Spacing:

Controlling line spacing improves readability and ensures uniformity across the document. Line spacing can be adjusted to meet the specified guidelines, such as single-spacing or double-spacing.

5. Page Breaks and Section Breaks:

Inserting page breaks between sections or chapters maintains the logical flow of the document. Section breaks are employed to apply different formatting styles within the same document.

6. Font Styles and Sizes:

Consistent font styles (e.g., Times New Roman, Arial) and sizes (e.g., 12pt) contribute to the document's readability and adherence to formatting standards.

Step 6: Saving the Formatted Document in a Database and Providing User Access

Once the document has undergone the formatting process, the final step involves storing the formatted version in the application's database, ensuring it is securely archived and easily accessible. This archival serves as a safeguard, preserving the document for future use and reference.

1.4 Architecture diagram

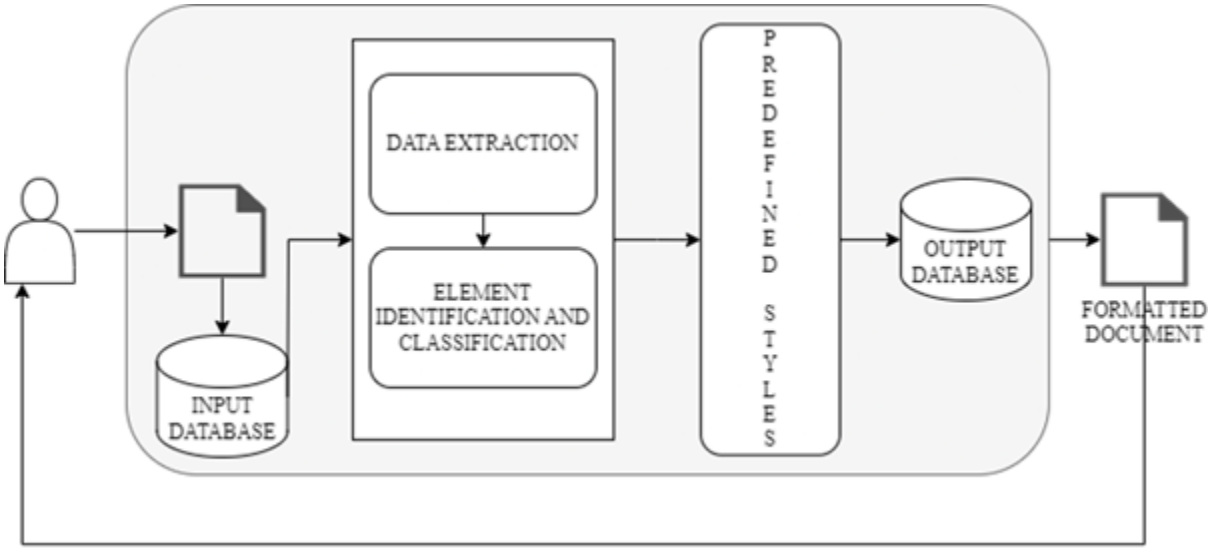

Figure 1.6 Architecture diagram

The structural design of the Automated Word Formatter ensures a fluid interaction between users and the underlying system. Users initiate the process by uploading Microsoft Word documents through the platform's user-friendly interface. These documents are securely stored in a MongoDB database, chosen for its adaptability and scalability, ensuring efficient document storage and retrieval. Once retrieved from MongoDB, the system engages in a detailed data extraction process facilitated by Python's python-docx module. This phase involves parsing and capturing both textual content and structural elements embedded within the document. Following this, an element identification stage takes place to discern between headings and non-headings, a crucial aspect for accurate formatting.

Machine learning becomes instrumental with the application of an XGBoost classifier for element classification. This dynamic approach ensures the differentiation of document components, laying the foundation for implementing predefined formatting styles. Each element is precisely formatted in accordance with established research paper standards. The now correctly formatted document seamlessly integrates back into the MongoDB database. This seamless cycle not only guarantees users access to the formatted document at their convenience but also sets the stage for generating a personalized download link.

The provision of a download link is a pivotal feature, directly connecting users to their formatted documents stored in the MongoDB database. This link acts as a bridge, allowing users to effortlessly retrieve their documents with a simple click. The overall result is an

efficient, user-centric process that aligns with the platform's commitment to delivering accurate and visually appealing document formatting.

1.5 Organization of the Report

This report consists of the overview of what all topics discussed in this entire report in a brief and concise manner with the sequence of topics presented.

Chapter 1: Introduction

This section discusses about the project and the use case of the project and how it is useful to the users and includes about the basic working of the overall project work flow.

Chapter 2: Literature Survey

This section discusses about the existing approaches to solve this problem and their drawbacks and the advantages. This section provides the required knowledge and a momentum to carry out the project.

Chapter 3: Proposed Methods

In this section we discussed about the logical sequence in which the problem is addressed and the methods that are adopted to solve the problem.

Chapter 4: Results and Discussions

This section contains information about the results obtained during the working of this project it contains the landing page an example of unformatted to formatted document and the step by step working results of the project.

Chapter 5: Conclusion and Future Enhancements

This section contains a robust solution for automating document formatting, ensuring precise style updates, and maintaining adherence to the highest standards. Users benefit from increased efficiency and accuracy in formatting, leading to time savings and elevated document quality. Additionally, it explores future aspects, providing a foundation for continual improvement and adaptation to evolving user needs.

CHAPTER 2

LITERATURE SURVEY

2.1 Existing Approaches

Nail Nasyrov and his team have introduced an innovative approach [1] for document formatting verification. In their work, they employ a decision tree algorithm, leveraging the efficiency of CatBoost. This algorithm meticulously classifies each element within documents as either syntactically correct or incorrect. The classification is based on a multitude of attributes, including but not limited to font size, alignment, line spacing, and more. The approach offers a systematic way to enhance the accuracy of document formatting verification, showcasing the team's commitment to advancing the field.

[2] Dr. Venkatesan, along with a collaborative team, introduces a template matching technique designed for comparing the layouts of two documents. This innovative method efficiently identifies differences between documents and subsequently adjusts them to align with the original document. The authors leverage the Python docx module for document manipulation, employ the difflib library to compare documents and pinpoint differences, and use regex to identify and handle specific patterns. This collaborative effort, led by Dr. Venkatesan V, showcases a systematic approach to document layout comparison and adjustment, highlighting the team's collective expertise in document analysis and manipulation.

In his work [3], Kirill Chuvilin introduces an innovative approach to LaTeX document processing, leveraging parse trees to represent document structures. The methodology involves training a machine learning algorithm on a carefully curated dataset of LaTeX documents, manually corrected by experts to include various errors such as typos, grammatical issues, and formatting errors. The algorithm, informed by the parse tree representation, adeptly identifies and rectifies these errors, offering a systematic and machine learning-driven strategy to enhance the precision and quality of LaTeX documents. This work contributes significantly to the field of document processing and error correction within the LaTeX framework.

In [4], Oliveira presents an approach employing two algorithms to automate document page layout. The first algorithm operates under the assumption that previously defined rectangular items are to be placed freely on the document. It calculates the minimum bounding box for each item, facilitating their placement. The second algorithm addresses the

placement of free-form items on pages divided into columns. This is achieved by determining the optimal number of columns for the page. Oliveira's approach offers a systematic and algorithmic solution to automate document page layout, contributing to advancements in document design and formatting.

Nathan Hurst presents a comprehensive survey in [5], exploring various existing approaches for automating page layout. The survey delves into different layout modes, including Coordinate, Flow, Grid, VH-Box, Guillotine, and Box, each of which governs how elements are positioned within documents. Additionally, the survey discusses approaches like Column-driven layout, Cell-driven layout, and Minimal configurations, which play crucial roles in selecting appropriate page layouts to tackle issues related to both micro and macro typography. Hurst's work provides valuable insights into the diverse methods available for automating page layout, contributing to the broader understanding of document design and formatting.

Nasser introduces a novel approach in [6] leveraging the Django framework, this method involves submitting a document as input. The document's contents are stored in a MySQL database and later processed to generate a new document with accurate formatting. Nasser's work stands out for its innovative use of web-based automation, incorporating Django and database management to streamline the word formatting process effectively. Nasser presents an innovative approach that leverages the Django framework to automate word formatting tasks. This method involves submitting a document as input through an interactive web interface. The contents of the document are then stored in a MySQL database and processed to generate a new document with accurate formatting. Nasser's work is notable for its efficient use of web-based automation, utilizing Django for web development and MySQL for database management to streamline the word formatting process effectively. Nasser's approach is implemented using the Django framework and MySQL database, allowing users to interact with the system through a user-friendly web interface. While this method currently provides templates tailored for specific student needs, its use of web technologies makes it convenient for users to input documents and receive formatted outputs seamlessly. By integrating web-based automation with database management, Nasser's approach demonstrates a novel way to automate word formatting tasks, enhancing efficiency and usability for document processing applications.

Gange delves into the optimization of table layouts in [7], offering three distinctive approaches for determining the optimal minimum height layout. These methodologies include the AI-based A* algorithm, Constraint Programming Approach 1, and Constraint Programming

Approach 2 with Lazy Clause Generation. The choice of the most effective method is contingent upon the unique characteristics of the tables in question, culminating in the adoption of a hybrid CP/SAT approach. Gange's contribution provides valuable insights and a diverse set of methodologies for enhancing table formatting. Gange explores the optimization of table layouts by presenting three distinct methodologies to determine the optimal minimum height layout for tables. These methodologies include the AI-based A* algorithm, Constraint Programming Approach 1, and Constraint Programming Approach 2 with Lazy Clause Generation. The choice of the most effective method depends on the specific characteristics of the tables being analyzed, leading to the adoption of a hybrid Constraint Programming (CP) and Satisfiability (SAT) approach for optimal table formatting. Gange's contribution offers valuable insights into diverse methodologies aimed at enhancing table layout optimization. Gange's work employs the A* algorithm and Constraint Programming approaches to optimize table layouts, focusing on minimizing the height of table structures. While these methodologies provide effective solutions, the paper acknowledges limitations such as not addressing nested tables, which are tables embedded inside other tables, leading to more complex layouts. Additionally, Gange's approach introduces a heuristic based on the area of cell content, which improves accuracy compared to previous heuristics used in similar contexts. Overall, Gange's research contributes to advancing table formatting techniques and offers practical methodologies for optimizing table layouts in various document processing applications.

Bilauca explores distinct approaches in [8] employing mixed-integer programming (MIP) and constraint programming (CP). The MIP model, while a more general and potent approach, may pose computational challenges, especially for large tables. In contrast, the CP model, while less general, often proves more efficient in solving real-world table layout problems. Through comprehensive evaluations on various real-world tasks, the authors demonstrate the system's capability to generate output comparable to human-generated results. Bilauca's work underscores the effectiveness of combining MIP and CP approaches for achieving optimal table layouts.

In [9], John Bateman provides a comprehensive exploration of Rhetorical Structure Theory (RST), a framework designed to elucidate the organization of text and the relationships existing between its constituent parts. RST operates on the fundamental premise that texts comprise two essential units: elementary discourse units (EDUs) and rhetorical relations. EDUs, representing the smallest analysable units of text within the RST framework, serve as the building blocks for understanding textual structure. Concurrently, the theory focuses on delineating rhetorical relations, which encapsulate the intricate relationships that bind these

EDUs. By elucidating the nuanced interplay between text elements, Bateman's work significantly contributes to our understanding of how textual content is organized and interconnected within the framework of Rhetorical Structure Theory.

In [10], Bolshakov introduces an inventive approach that employs a mathematical cohesion comparator function to evaluate each line within a document. The ensuing cohesion scores are systematically compared, and paragraph segmentation is determined based on a precomputed threshold. The decision to split paragraphs hinges on assessing the disparity in cohesion scores between two lines against the established threshold. This threshold is meticulously derived through an in-depth analysis of a database comprising collocations and semantic rules. Bolshakov's methodology, grounded in mathematical cohesion analysis and semantic rules, offers an analytical and data-driven approach to enhance paragraph segmentation, contributing to the evolution of document organization and readability.

TableNet, developed by Shubham Singh [11], introduces an innovative deep learning model tailored for comprehensive table detection and structured data extraction from scanned document images. Utilizing a two-stream encoder-decoder architecture, the model's encoder extracts features from input images, while the decoder generates two vital output masks—one for the table region and another for individual columns within the table. Addressing the challenges posed by unstructured document images containing tables, TableNet goes beyond traditional table detection. It intricately tackles the task of extracting information from rows and columns within the detected tables. The model's holistic approach, emphasizing both accurate table identification and nuanced structure recognition, responds to the increasing demand for efficient information extraction from diverse documents captured through mobile phones and scanners.

In [12], Widiastuti addresses the formidable task of named entity recognition and classification in historical documents, a challenge amplified by document variety, quality variations, and the scarcity of labelled data. Both rule-based and machine learning approaches have been proposed to tackle these challenges, with recent strides in deep learning and domain-specific systems offering promising avenues for advancement. The digitization of historical documents has ushered in a new era of accessibility, but it also presents challenges in efficiently mining information from this vast repository. In the abstract, Widiastuti underscores the significance of developing technologies to search, retrieve, and explore information within this "big data of the past." The focus on named entity recognition (NER) systems in the survey aligns with the demands of humanities scholars, highlighting the challenges posed by diverse, historical, and noisy inputs. The survey not only inventories existing resources and outlines

past approaches but also outlines key priorities for future developments in this crucial area of historical document analysis.

In [13], Stefan Schweter introduces FLERT, a pioneering method that enriches Named Entity Recognition (NER) models by incorporating document-level features through a Convolutional Neural Network (CNN). FLERT systematically extracts two types of document-level features: global features, encapsulating the overall characteristics of the document, and local features, capturing relationships between named entities within the document. These features are seamlessly integrated into a standard NER model, contributing to enhanced performance. In the broader context of NER approaches traditionally focusing on sentence-level information, FLERT distinguishes itself by leveraging transformer-based models to naturally capture document-level features. The paper conducts a comparative evaluation within the standard NER architectures of "fine-tuning" and "feature-based LSTM-CRF," exploring different hyperparameters for document-level features. Valuable insights from experiments lead to recommendations on effectively modeling document context. The approach is integrated into the Flair framework for reproducibility, presenting new state-of-the-art scores on several CoNLL-03 benchmark datasets and reaffirming the effectiveness of FLERT in advancing document-level feature incorporation for improved NER performance.

In [14], Ming Zhou and their team introduce LayoutLM, a pioneering pre-training framework for Document Image Understanding (DIU) that adeptly predicts both text and layout information in scanned document images. Through extensive pre-training on a large-scale dataset encompassing various tasks such as text recognition, layout analysis, and mask prediction, LayoutLM achieves a holistic understanding of scanned documents. Unlike previous NLP-focused pre-training models that predominantly emphasize text-level manipulation, LayoutLM uniquely incorporates layout and style information crucial for document image understanding. This innovative approach allows LayoutLM to excel in real-world tasks, including information extraction from scanned documents. Leveraging image features to integrate visual information into the model, LayoutLM represents a breakthrough as the first framework to jointly learn text and layout in a unified manner for document-level pre-training. The versatility of LayoutLM is demonstrated through state-of-the-art results in diverse downstream tasks, such as form understanding, receipt understanding, and document image classification. The code and pre-trained models are made publicly accessible, enhancing accessibility and fostering further advancements in the field.

In reference [15], Widiastuti presents a modular Document Image Extraction System designed to convert document images into editable text, enabling efficient storage and

retrieval of textual content. This system is particularly valuable for digitizing paper documents and making their contents accessible in digital formats. The key innovation of this system lies in its modular architecture, which integrates diverse text extraction methods to overcome inherent challenges associated with document image processing. The modular design allows for flexibility and adaptability, enabling the system to incorporate various techniques based on specific document characteristics and quality.

In reference [16], by Firoozeh, delve into the domain of automatic keyword extraction, a critical technique for identifying the most significant words and phrases within text. The paper explores and compares three distinct approaches for keyword extraction: statistical methods, machine learning methods, and hybrid methods that combine elements of both. The adoption of statistical methods involves analyzing the frequency and distribution of words in a given text corpus to identify keywords based on their occurrence patterns. Machine learning methods, on the other hand, leverage models trained on annotated datasets to predict and extract keywords autonomously from text inputs. Hybrid methods integrate aspects of statistical and machine learning techniques to achieve more robust and accurate keyword extraction outcomes. The importance of effective keyword extraction is highlighted by its utility across diverse domains, including news articles, scientific papers, and social media posts. By automatically identifying relevant keywords, these techniques enable applications to efficiently process and categorize textual content from various sources.

In reference [17], Tejas presents a method comprising five essential steps for table extraction from complex documents. The process begins with document preprocessing to prepare the input document, followed by table detection to identify potential table regions. Subsequently, table structure recognition analyzes the layout and organization of table components, leading to content extraction where text or data within table cells is retrieved. Tejas's approach acknowledges challenges posed by complex table structures, noting that traditional methods often struggle with nested rows, merged cells, and irregular layouts. Additionally, reference [17] by T. Kashinath et al. explores end-to-end table structure recognition and extraction in heterogeneous documents. This work highlights the advantages and challenges of employing end-to-end methods, leveraging advanced techniques like deep learning for accurate table extraction, particularly beneficial for handling complex table structures effectively

This is an article about a dataset for document layout analysis [18]. It discusses how current document layout analysis methods rely on computer vision and not text [18]. The

authors created a dataset called DocBank that includes images and text together [18]. DocBank will allow researchers to develop better document layout analysis methods [18]. DocBank is available to the public [18]. M. Li and colleagues introduced DocBank, a benchmark dataset designed to enhance document layout analysis methods. Unlike traditional approaches that mainly rely on computer vision, DocBank uniquely combines images and text within its dataset. This integration enables researchers to develop more sophisticated document layout analysis algorithms, leveraging both visual and textual information for improved accuracy and robustness. DocBank is the largest dataset of its kind, containing over 500,000 document pages with fine-grained token-level annotations. Researchers can use DocBank to advance tasks such as text block detection, text line extraction, and logical structure analysis, driving progress in document processing and information extraction methods. More details about DocBank can be found in the original paper by M. Li.

In [19] Stefan research paper proposes an unsupervised approach for analysing the structural layout of digital scientific articles. The method leverages advanced clustering techniques including hierarchical clustering and k-means clustering to group words, lines, and text blocks within the document. By applying clustering algorithms, the approach aims to identify meaningful patterns and relationships in the document's content and layout. In addition to clustering, the research incorporates geometric analysis to understand the spatial arrangement of text blocks. This geometric analysis helps in deciphering the logical and hierarchical connections between different sections and components of the document. By considering both content-based clustering and geometric relationships, the accuracy of identifying document structure is improved. Despite the promising advantages of this approach, Stefan's paper acknowledges certain limitations. One key limitation is the dependence on well-structured document formats. The effectiveness of the method may vary depending on how well the scientific articles adhere to standard formatting practices. Furthermore, the paper notes the absence of robust error handling mechanisms, which could affect the reliability of the structural analysis, especially in the presence of noisy or poorly formatted documents.

In [20], the authors present a method for automatically analyzing the structure of structured PDF files, tackling the challenges associated with information extraction from such documents. The system utilizes a Support Vector Machine (SVM) classifier to categorize elements within the PDF, classifying them into distinct structural components like paragraphs, subsections, headings, and other relevant categories. Following classification, the system then

organizes the extracted information into an ontological representation. This ontological representation serves as a structured framework that captures the relationships and hierarchy between different document elements, facilitating efficient information retrieval and analysis. The approach outlined in [20] offers a systematic solution for structuring and understanding content within structured PDF files, leveraging machine learning techniques like SVM classification and ontological representation for effective document analysis.

In [21] a novel hybrid Named Entity Recognition (NER) system specifically designed for legal documents is adopted. The system adopts a transfer learning approach by integrating pre-trained spaCy models, namely "en_core_web_trf" and "en_core_sci_sm", to leverage their general language understanding capabilities [21]. These pre-trained models serve as a foundational knowledge base for recognizing named entities within text. To enhance accuracy in the legal domain, the system fine-tunes these pre-trained models using domain-specific rules tailored for legal entity recognition. By incorporating legal-specific rules into the transfer learning process, the system adapts the pre-existing language understanding to better identify and classify legal entities such as laws, regulations, organizations, and entities pertinent to legal contexts. A key advantage of this approach is its reduced reliance on large manually annotated datasets, which are often scarce to create in the context of NER tasks. Leveraging transfer learning and domain-specific fine-tuning enables the system to achieve improved performance in legal entity recognition while mitigating the need for extensive manual data annotation.

This is an article about evaluating a method for identifying sections in clinical documents [22]. It discusses how clinical notes are divided into sections with headers. These sections can be labeled or unlabeled. The authors designed an algorithm called SecTag to identify these sections [22]. SecTag uses a combination of techniques, including natural language processing, to find section headers [22]. The study evaluated SecTag’s performance on a set of clinical documents. The results showed that SecTag could accurately identify both labeled and unlabeled sections [22]. the authors evaluate a method for identifying sections within clinical documents, where notes are typically organized into sections with headers. The study focuses on the challenge of accurately identifying both labeled and unlabeled sections within these documents. The authors developed an algorithm named SecTag, which utilizes a combination of natural language processing techniques to detect section headers. This

algorithm was specifically designed to address the unique structure and variability of clinical notes. The evaluation of SecTag's performance involved assessing its ability to accurately identify section headers in a set of clinical documents. The results demonstrated that SecTag performed well in identifying both labeled and unlabeled sections, particularly within history and physical documents. Despite encountering some challenges with the accuracy of identifying unlabeled sections, the SecTag algorithm, supplemented with naive Bayesian scoring methods, showed promising performance in automating section identification within clinical documentation. This study highlights the potential of NLP-based approaches like SecTag in improving the efficiency and accuracy of clinical document analysis and organization.

In [23], the authors introduce a document clustering method aimed at classifying documents based on their structural characteristics. This approach utilizes the Support Vector Machine (SVM) tool LIBLINEAR, which is specifically designed for large-scale text classification tasks, to efficiently group documents into three distinct categories: strong structure, weak structure, and unstructured. The system analyzes patterns within the documents to perform this classification, leveraging the capabilities of LIBLINEAR to handle the classification of documents based on their structural attributes. The clustering method employed in this approach involves using LIBLINEAR to categorize documents into three sections based on their identified structural characteristics: strong structure, weak structure, and no structure. While this method demonstrates effectiveness, there are opportunities to enhance performance and efficiency further by leveraging advanced algorithms such as Graph Neural Networks (GNNs). These advanced techniques could potentially improve the accuracy and robustness of the document clustering process by incorporating more complex patterns and relationships identified through NLP preprocessing steps. Overall, this approach highlights the potential of leveraging machine learning tools and NLP techniques for document structure analysis and classification, offering insights into document organization and content understanding.

In [24], the authors present an innovative approach to automatically analyze the structure of structured PDF files, addressing the challenges associated with extracting information from unstructured documents. The proposed solution involves an intelligent method to identify and recognize the layout and structure of these PDF files, including the identification of paragraphs, tables, and associated text content. The extracted information is organized and stored in an ontology, providing a structured representation of the document's

content. The authors tested their approach on a collection of construction tender documents, showcasing its effectiveness in real-world applications. This collection comprises a substantial amount of annotated documents, which serves as a valuable resource for layout analysis tasks. However, it's noted that the automatically generated annotations may contain errors, potentially affecting the performance of overfit models. Despite this, the dataset consists of 360,000 document images and over 1 million document annotations, making it a comprehensive and significant dataset for advancing research in document layout analysis and information extraction. This work demonstrates the potential of automated methods for document structure analysis, leveraging large annotated datasets to develop and evaluate techniques for efficiently processing and understanding structured PDF documents. For more details on this study, readers can refer to the original article referenced as [24].

The paper by J. M. Ponte, titled "Text segmentation by topic," introduces an innovative method for identifying topics within a document through text segmentation techniques. The goal is to automatically organize and cluster text based on thematic relevance. Firstly, the document undergoes pre-processing steps aimed at removing noise and irrelevant elements that could hinder topic identification. This includes tasks such as removing stop words, punctuation, and other non-essential elements from the text. The paper introduces a method to expand keywords by incorporating synonyms and related terms, thereby enriching the vocabulary associated with the document's content [25]. This expanded vocabulary is instrumental in capturing a broader range of semantic concepts and associations related to the document's topics.

Table 2.1 Representation of Summary of Existing Approaches

Ref. No	Methodology	Drawbacks	Advantages
[1]	It is a classification technique which uses a decision tree algorithm to classify elements of	It only checks the errors in documents, but it doesn’t edit the document automatically.	Utilizes machine learning algorithms, specifically gradient boosting on

	document there by applying the predefined style to the document.		decision trees improving performance.
[2]	This approach is based on the usage of docx, regex and difflib library's present in python which are used to classify every element as either correct or wrong.	Lack of template customization options and does not predict mathematical equations.	Provides user friendly experience and reducing the risk of document rejection and improving overall document quality.
[3]	This method uses groups with a tree pattern and group rules to identify potential errors.	Can wrongly identify a correct statement to avoid such errors a large number of computational resources are required	It uses Zhang-Shasha algorithm to Correct a wide range of errors including syntax errors, style errors, and formatting errors.
[4]	The first approach uses a bounding box method. The second approach is a places item in free form	The algorithms are highly dependent on the input quality of the document.The approach doesn't provide good results on tables and image data.	Recursively divides the document into pages. Places the document items on the pages according to a set of heuristics there by pruning entire traversal
[5]	This survey explains about techniques such as Column-driven layout, Cell-driven layout selection methods	It does not use any machine learning approach to automate document formatting it uses the tree structure of the document to edit formatting	It provides accurate formatting as this approach modifies the tree structure of the document.
[6]	Implemented using Django framework and MySQL Database.	It only provides template for a specific student template.	Interactive web interface making it convenient for users.

[7]	It uses A* algorithm approach Constraint Programming Approach 1 Constraint Programming Approa ch 2 with Lazy Clause Generation	Drawback of the paper is that it does not address the problem of nested tables. Nested tables are tables that are embedded inside other tables which can be used to create complex layouts	This heuristic used in this approach is based on the area of the cell content and is more accurate than the previously used heuristics.
[8]	It used two different approaches: MIP (Mixed Integer Programming) and CP (Constraint Programming)	Automatic table layout algorithms can sometimes generate tables that are not accurate or consistent. This is especially true for tables with complex layouts or tables that contain a lot of data.	It generates tables in a variety of different formats, such as HTML, PDF, and CSV.
[9]	It uses NLP techniques such as chunking parts of speech tagging and semantic role labelling and statistical model Hidden Markov Models (HMM's)	The system is not able to generate diagrams that are interactive and dynamic.	Generates a variety of different types of output including news articles, scientific papers, and technical documentation.
[10]	It is achieved in following steps: Quantitative evaluation of text cohesion, Smoothing and Normalizing the cohesion function, Splitting text into paragraphs.	Precision value is low as compared to other experts.	The cohesion function is constructed basing on close co-occurring at words pairs contained in a large database collection.
[11]	TableNet is a two-stream encoder-decoder architecture that extracts features from the input image and generates two output masks, one for the table region and one for the column region.	TableNet may not be able to accurately detect and extract tabular data from complex tables, such as tables with nested headers, merged cells, or irregular structures.	TableNet is a deep learning model, which means that it is able to learn complex patterns from the data. This makes it more robust to noise and variations in the

			appearance of tables in scanned document images.
[12]	NERC in historical documents uses both rule-based and machine learning approaches, with recent advances in deep learning and domain-specific systems.	NERC models require a large amount of labelled data to train effectively. This data can be difficult and expensive to collect, especially for historical documents.	NERC models can be trained to achieve high accuracy in identifying and classifying named entities in historical documents. This is important for many applications, such as digital humanities research and historical archiving.
[13]	FLERT: Document-Level Features for Named Entity Recognition, FLERT allows NER models to capture document-level information that is not available to traditional NER models.	FLERT requires a large amount of labelled data to train effectively. This data can be difficult and expensive to collect, especially for historical documents and low-resource languages.	FLERT has been shown to improve the accuracy of NER models on a variety of datasets. This is because FLERT allows NER models to capture document-level information that is not available to traditional NER models.
[14]	LayoutLM is pre-trained on a large-scale dataset of scanned document image. Once the model is pre-trained, it can be fine-tuned on a variety of DIU tasks	If the pre-training data is biased, the model will be biased as well. This can lead to errors when the model is used on real-world data.	It is pre-trained on a large-scale dataset of scanned document images.
[15]	It Contains 6 components: Document image acquisition, Pre-processing, Text extraction, Feature extraction, Extraction	The system requires a large amount of training data to be trained effectively. This data can be difficult and expensive to collect.	The system is able to extract text from a variety of document types with high accuracy. This is important for applications where the extracted

			information needs to be reliable.
[16]	Uses three different approaches: Statistical methods, Machine learning methods, Hybrid Methods	misspellings and grammatical errors. This can lead to the extraction of incorrect keywords.	This is important for applications that need to process texts from a variety of domains, such as news articles, scientific papers, and social media posts
[17]	Table extraction is done in 5 steps: Document image pre-processing, Table detection, Table structure recognition, Table content extraction.	End-to-end methods may have difficulty handling complex table structures, such as tables with nested rows and columns, and merged cells.	End-to-end methods can achieve higher accuracy than traditional methods, especially on complex or challenging documents.
[18]	This dataset can be used to meet a wide variety of applications such as Text block detection , Text line extraction , Logical structure analysis.	Can include more number of documents data.	DocBank is the largest dataset of its kind, with 500K document pages. DocBank provides fine-grained token-level annotations for layout analysis.
[19]	This includes hierarchical agglomerative clustering (HAC) and k-means clustering to merge characters into words, lines, and blocks. It also uses geometric relations between text blocks.	Lack proper structure, the system's performance may degrade. Does use any error handling methods.	Geometric relationship extraction allows for precise and quantitative analysis of spatial information. HAC can handle noisy data by forming them into singleton clusters or small branches in

			the dendrogram. This helps in handling noisy data.
[20]	Text line extraction, Text block grouping, Font features and layout feature extraction. Uses SVM classifier to classify elements as paragraph subsection etc.	It lacks user interface it does not handle unstructured pdf documents.	Handles more complex structured pdfs. The approach is language-independent, meaning that it can be used to analyze PDF files in any language.
[21]	Used a hybrid approach by integrating two spacy library's prebuilt NER models en_core_web_trf and en_core_sci_sm by specifying unique rules.	Annotating a large amount of data to create a dataset manually is tedious whereas a smaller dataset give a lower accuracy.	It uses transfer learning approach in which the pre-built spacy models are trained on huge datasets and then fine tuning them on a specific approach.
[22]	The author used a combination of SecTag algorithm and naive Bayesian scoring methods to identify note section headers.	The identification of the unlabeled sections gave a lower accuracy score.	The SecTag algorithm acc urately identified both labeled and unlabeled sections in history and physical documents.
[23]	Used a linear SVM tool LIBLINEAR clustering method which groups documents into three sections namely strong structure weak structure and no structure.	Advanced algorithms such as Graph neural networks can be used to enhance the performance and efficiency of the approach.	Identifies the structure of a document by using various NLP preprocessing steps and there by clustering the documents.
[24]	It is a collection of huge amount of annotated documents which can be used for layout analysis.	The automatically generated annotations may have errors which can significantly affect the performance of an overfit model.	It consists of 360,000 docu ment images and over 1 million document annotati

			ons. The annotations are automatically generated
[25]	It uses hierarchical clustering algorithm. The algorithm works by iteratively merging the two most similar clusters until a desired number of clusters is reached.	It uses supervised learning approach adopting to an unsupervised approach provides a huge mass of data on which a model can be trained.	It uses many NLP techniques such as removal of stop words conversion of words to a vector and usage of attention based mechanism in identification of word dependencies.

CHAPTER-3

Proposed Method

3.1 Problem Statement & Objectives of the Project

Problem Statement: Transforming Document Formatting

In the fast-paced world of academia and research, scholars often grapple with the intricacies of document formatting. Whether it's adhering to specific style guidelines, ensuring uniformity across sections, or grappling with the nuances of various formatting styles, the process can be time-consuming and error-prone. The existing tools and platforms fall short in providing a seamless solution to this persistent challenge. Manual formatting consumes valuable time that could be better utilized for research and content creation. Inconsistencies in formatting may lead to a lack of professionalism in academic and research documents, affecting the overall quality and impact of scholarly work.

The need for an efficient, automated document formatting tool is evident. This tool must not only align with the rigorous standards of academic formatting but also be intuitive, user-friendly, and adaptable to diverse formatting requirements. Addressing this need will empower researchers, students, and academics to focus more on the substance of their work rather than getting bogged down by formatting complexities. In the realm of academia and research, scholars face a persistent challenge when it comes to document formatting. The meticulous adherence to specific style guidelines, achieving uniformity across different sections, and navigating through the nuances of diverse formatting styles can be both time-consuming and prone to errors. Current tools and platforms available often fall short of providing a seamless solution to this dilemma. Manual formatting processes consume valuable time that could otherwise be allocated to research and content creation.

Moreover, inconsistencies in formatting may compromise the professionalism of academic and research documents, ultimately impacting the quality and credibility of scholarly work. The necessity for an efficient and automated document formatting tool is clear. Such a tool must not only meet the rigorous standards of academic formatting but also be intuitive, user-friendly, and adaptable to a wide range of formatting requirements. By addressing this need, researchers, students, and academics would be empowered to direct more focus towards

the substance and essence of their work rather than being encumbered by the complexities of formatting tasks. The challenge lies in developing a comprehensive solution that not only automates formatting procedures but also anticipates and meets user expectations with flexibility. This tool should evolve alongside the dynamic landscape of document standards, bridging the gap between intricate formatting requirements and streamlined scholarly writing processes. Ultimately, the aim is to transform document formatting into a seamless and efficient aspect of scholarly writing, thereby enhancing productivity and allowing individuals to devote more time and energy to the core aspects of their research and academic pursuits.

Objective of the Project

To Implement a python-based word formatting tool which automates manual editing to save time and effort.

The approach used is to classify the elements of the document this process is carried out by analyzing the xml structure of the document.

To implement approaches for formatting text, images, and tables and configuring the type of text and its style by using a XG Boost classifier model.

3.2 Architecture Diagram

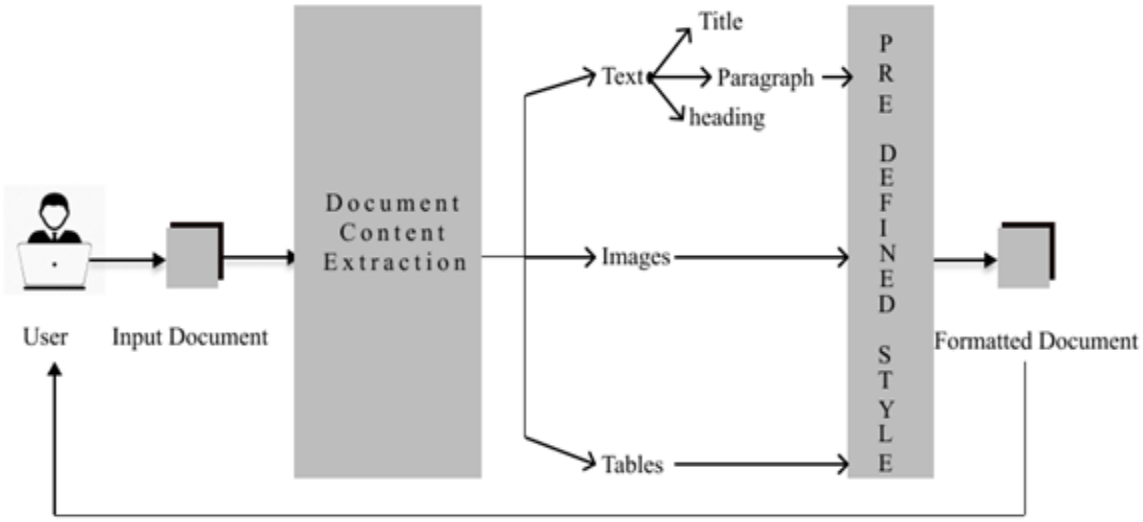

Figure 3.1 Architecture Diagram

As illustrate in fig 3.1, the architectural design is a more detail view of the Automated Word Formatter, it organises a smooth interaction between users and the underlying

system. Users initiate the process by uploading Microsoft Word documents through the platform's intuitive interface. These documents find their repository in a NoSQL MongoDB database, as a document is unstructured type of data making it a perfect choice to select a NoSQL database, additionally its adaptability and scalability, ensuring efficient document storage and retrieval are the key features provided by MongoDB. Once the document is retrieved from MongoDB, the system engages in a data extraction process facilitated by the python-docx module. This step involves parsing and capturing both textual content and structural elements embedded within the document. Subsequently, an element identification phase takes place to discern between headings and non-headings, a pivotal aspect for accurate formatting.

Following the data is extracted Machine learning comes into play with the application of an XG Boost classifier for element classification which involves classifying the elements as either headers, non-headers, tables or images. This dynamic approach ensures the differentiation of document components, paving the way for the implementation of predefined formatting styles. Each element is formatted with precision in accordance with established research paper standards. Then each element is now analysed using the regex module to capture its style once these style attributes are captured, the system automatically updates the style of each element accordingly. This ensures a precise adjustment of the document's appearance, aligning it with the predefined formatting standards. The use of regex adds a sophisticated layer to the formatting process, making it adaptive to diverse document styles and enhancing the overall accuracy of the project.

The correctly formatted document is then seamlessly reintegrated into the MongoDB database. This harmonious cycle not only ensures the user's ability to access the formatted document at their convenience but also sets the stage for the generation of a personalized download link. The provision of a download link is a key feature, directly connecting users to their formatted documents stored in the MongoDB database. This link acts as a bridge, enabling users to effortlessly retrieve their documents with a simple click. The end result is an efficient, user-centric process that aligns with the platform's commitment to delivering correct and aesthetically pleasing document formatting.

Major Components of Architecture diagram:

Efficient Document Upload and Storage

The process begins with users initiating document formatting by uploading Microsoft Word files through the platform's intuitive interface. These uploaded documents are

securely stored in a NoSQL MongoDB database, chosen for its adaptability and scalability in handling unstructured data like documents. MongoDB efficiently manages document storage and retrieval, ensuring that users can access their documents easily and quickly.

Intelligent Data Extraction and Element Identification

Once a document is retrieved from MongoDB, the system employs the python-docx module for data extraction. This step involves parsing both textual content and structural elements embedded within the document, preserving its layout and formatting. Subsequently, an element identification phase distinguishes between headings and non-headings, a crucial step for accurate formatting.

Machine Learning-driven Element Classification

The system leverages Machine Learning with an XGBoost classifier for element classification. This dynamic approach categorizes document elements into headers, non-headers, tables, or images, enabling precise differentiation of document components. This classification lays the foundation for applying predefined formatting styles tailored to each element.

Adaptive Formatting with Regex

Each document element undergoes analysis using the regex module to capture its style attributes. This information is then used to automatically update the style of each element, ensuring precise adjustments to the document's appearance according to established formatting standards. The use of regex adds sophistication to the formatting process, making it adaptable to diverse document styles and enhancing overall accuracy.

Seamless Integration and Personalized Document Retrieval

The correctly formatted document seamlessly reintegrates into the MongoDB database, ensuring users can access their formatted documents conveniently. The system generates personalized download links that directly connect users to their documents stored in MongoDB. This feature enables effortless document retrieval with a simple click, enhancing user experience and accessibility.

Automated Document Formatting Pipeline

The system orchestrates an automated document formatting pipeline that streamlines the entire process from upload to retrieval. This pipeline encompasses sequential stages of document processing, including extraction, identification, classification, formatting, and storage. Each stage is interconnected to ensure a cohesive and efficient workflow for transforming raw document data into formatted outputs.

Scalability and Performance Optimization

The architecture prioritizes scalability and performance optimization to accommodate varying user demands and document processing requirements. Implementations such as load balancing, distributed computing, and caching strategies enhance system responsiveness and scalability. The system is designed to handle large volumes of document uploads and formatting tasks efficiently, ensuring consistent performance under varying workloads.

User Feedback and Analytics Integration

To continuously improve the document formatting experience, the system integrates user feedback mechanisms and analytics tools. User feedback on formatting quality and preferences informs iterative improvements to the system's algorithms and user interface. Analytics data provides insights into usage patterns, document processing trends, and performance metrics, enabling data-driven enhancements and optimizations over time.

Security and Access Control Measures

Security measures are integrated into the architecture to safeguard user data and documents. This includes encryption protocols for data transmission and storage, access control mechanisms to manage user permissions, and auditing features to monitor system activities. Compliance with data protection regulations ensures the confidentiality and integrity of user information throughout the document formatting process.

3.3 Software and Hardware Requirements

These generalized software and hardware requirements need to be satisfied in order to build this Automated Word Formatter.

Frontend Development	**Software:** Text Editor (e.g., Visual Studio Code, Atom, Sublime Text) and a Web Browser (e.g., Google Chrome, Mozilla Firefox) **Hardware:** Computer with a minimum of 4GB RAM and High-resolution display.

Backend Development	**Software:** Python: Install the latest version of Python >=3.8.0, Install the Django web framework and scikit-learn machine learning module supported by python. **Hardware:** Computer with a minimum of 8GB RAM and Adequate free storage >= 10gb to store project documents and database.
Database	**Software:** MongoDB Compass and MongoDB Atlas account to access the cloud-based MongoDB. **Hardware:** free storage >= 1gb to store project documents that are fetched from the database.

Additional Requirements

Machine Learning Framework: While XGBoost is used in this example, other machine learning frameworks like TensorFlow or PyTorch could be explored depending on the desired level of customization and future feature development.

Regular Expression Library: The regex module is used for advanced style matching and manipulation. While optional, it can enhance the AWF's adaptability to diverse document styles.

3.4 Modules and its Description

3.4.1 User Module

Registration and Authentication:

User Registration: This involves providing a mechanism for users to create accounts within the system. During registration, users typically input essential information such as their username, email address, and password.

Authentication: Once registered, users need a secure way to log into their accounts. Authentication ensures that only authorized users can access the system. This process often involves verifying a user's identity through credentials (e.g., username and password) and employing secure authentication protocols to prevent unauthorized

User Profile Management

Profile Updates: Users should have the ability to manage and update their profiles. This includes modifying personal information like names, email addresses, and passwords. A user-friendly interface is essential to make this process straightforward.

Security Measures: Robust security measures, such as password encryption and multi-factor authentication, are implemented to safeguard user information.

File Upload and Download

File Upload: This feature allows users to upload Word documents or other specified file types to the system. Proper validation and security checks are necessary to prevent malicious uploads and ensure the integrity of the uploaded files.

Download Functionality: Users is able to download formatted documents or files from the system. Access controls and permissions should be in place to regulate which users can download specific content.

3.4.2 Input and Processing Module

File Upload Handling

Receiving Files: This involves creating a mechanism to receive uploaded Word documents from users. The system should have a secure and reliable file upload component that can handle various file sizes and formats.

Validation: Before processing the uploaded files, it is crucial to implement validation checks to ensure that the files are of the expected type (Word documents, in this case) and do not contain any malicious content. This helps in maintaining the integrity of the system and preventing potential security vulnerabilities.

Storing in Database: Once the file is validated it is stored in the remote cloud-based MongoDB database a NoSQL database which supports un-structed data like the documents.

Data Extraction

When a Word document is uploaded to the system for data extraction, the process involves meticulously retrieving both textual content and the underlying document structure. Text extraction is fundamental, encompassing the parsing and retrieval of paragraphs, headings, captions, and other textual elements present in the document. This step requires identifying and isolating different types of text while preserving formatting details like font styles, sizes, and alignments. Additionally, special text elements such as footnotes, endnotes, and citations need to be recognized and captured to ensure comprehensive content extraction.

In parallel, understanding the document's structure is equally crucial for effective data extraction. Document structure identification entails recognizing and categorizing various

layout-related components within the Word document. This includes detecting sections, headers, footers, tables, lists, and other organizational elements that contribute to the document's presentation and organization. By accurately identifying and categorizing structural elements, the system gains insights into the hierarchical layout of the document, paving the way for downstream processing tasks such as element classification and formatting application.The combination of text extraction and document structure identification forms the foundation of the data extraction process, transforming raw document data into structured and actionable information. This comprehensive approach ensures that the automated document processing system can efficiently handle diverse document formats while preserving content integrity and facilitating subsequent analysis and formatting tasks.

Data Preprocessing

Data preprocessing plays a critical role in refining and preparing extracted data from documents for further analysis and processing within automated systems. The initial extraction process often results in raw data that may contain unwanted elements such as extra spaces, special characters, or artifacts from formatting styles used in the original document. Cleaning the extracted data involves systematically removing these artifacts to ensure the accuracy and consistency of the information. This cleaning process not only enhances the quality of the data but also prepares it for subsequent stages of analysis and transformation.

In addition to cleaning, data preprocessing involves standardizing formats to ensure data consistency and compatibility with the system's requirements. This step may include converting extracted data into a common format or structure, such as normalizing dates, numbers, or text representations. Standardization facilitates seamless integration and analysis of data across different sources and helps in reducing inconsistencies that may arise from variations in document formatting. Furthermore, implementing quality assurance measures during data preprocessing is crucial to detect and address any anomalies or errors in the extracted data. Quality checks can involve validating data integrity, identifying incomplete or erroneous entries, and ensuring adherence to predefined data standards. By implementing robust quality assurance practices, the processed data becomes reliable and suitable for use in downstream applications such as machine learning models, data analytics, or document formatting tasks. Overall, data preprocessing acts as a foundational step in the document processing workflow, ensuring that the extracted data is refined, standardized, and ready for meaningful analysis and utilization within automated systems.

3.4.3 Predefined Style Application Module

Layout Adjustment:

Compliance with Standards: One of the primary functions of this module is to ensure that the layout of the document adheres to predefined standards, especially in the context of research papers. This involves adjusting elements such as margins, spacing, indentation, and page formatting to meet the specified requirements.

Template Application: In cases where a specific document template or layout is required, the module may apply or adjust the document structure to conform to the predefined template. This ensures consistency in the visual presentation of documents within a given context.

Style Consistency:

Textual Styles: Applying consistent styles to textual elements throughout the document is essential. This includes aspects such as font type, size, and color. The module may use regular expressions or other pattern-matching techniques to identify and modify text based on predefined style rules.

Headings and Subheadings: Ensuring consistent styling for headings and subheadings is crucial for document readability and professional presentation. The module can automatically detect and apply styles to these structural elements based on predefined criteria.

Citations and References: In academic or research documents, the module may handle the consistent application of citation styles (e.g., APA, MLA) and formatting of references. This ensures that citations and reference lists follow the prescribed rules and guidelines.

Dynamic Styling Rules:

Configurability: The module may allow for configurability to accommodate different style requirements based on user preferences or specific document types. This flexibility enables users to customize the application of styles to suit their needs.

Rule-Based Styling: Implementing a rule-based system allows the module to apply styles dynamically based on a set of predefined rules. This could involve recognizing certain patterns, keywords, or document structures and applying styles accordingly.

Version Control:

Document Versioning: If document version control is part of the system, the module may play a role in maintaining consistency across different versions of a document. This ensures that style changes or updates are applied uniformly throughout the document history.

3.4.4 Output Module

Formatted Document Generation:

Content Integration: The module integrates with other components, such as the Input and Processing Module and the Predefined Style Application Module, to gather processed and styled content. This may include text, images, and other media elements.

Document Assembly: Formatted document generation involves assembling the processed content into a cohesive and well-structured document. This includes arranging sections, applying styles, and ensuring that all elements are properly formatted according to the specified standards or styles.

Dynamic Content: In cases where the content is dynamic or user-specific, the module dynamically generates documents tailored to individual preferences or parameters.

File Download Handling:

User Interaction: The module provides a user-friendly mechanism for users to download the generated documents. This could involve the presentation of a download button or link within the user interface.

File Format Considerations: Depending on the system requirements, the Output Module may support various file formats. In this case, it specifically generates and handles Word documents. The module ensures that the generated document is in the appropriate format, compatible with common word processing software.

Download Permissions: Implementing download permissions and access controls is essential to regulate which users can download specific documents. This helps maintain security and confidentiality, especially in environments where document access needs to be restricted.

3.5 Requirements Engineering

3.5.1 Functional

File Upload: Users should be able to easily upload Word documents through the web interface.

Automatic Formatting: The system must automatically format the uploaded documents according to research paper standards.

Download Options: The application should offer users the option to download the formatted document in different formats (e.g., Word, PDF).

3.5.2 Non-Functional

Security and Privacy: Users' uploaded documents must be handled securely to maintain data privacy and prevent unauthorized access.

Cross-Browser Compatibility: The web application should work seamlessly across various web browsers (Chrome, Firefox, Safari, etc.) to accommodate user preferences.

Performance: Users expect quick and efficient processing of their documents without significant delays.

Clear Error Messages: In case of errors, users should receive clear and user-friendly error messages to understand and address issues.

3.6 Analysis and Design through UML

This analysis contains Class Diagram, Sequence Diagram, Use case diagram and the activity diagram of Automated Word Formatter.

3.6.1 Class Diagram

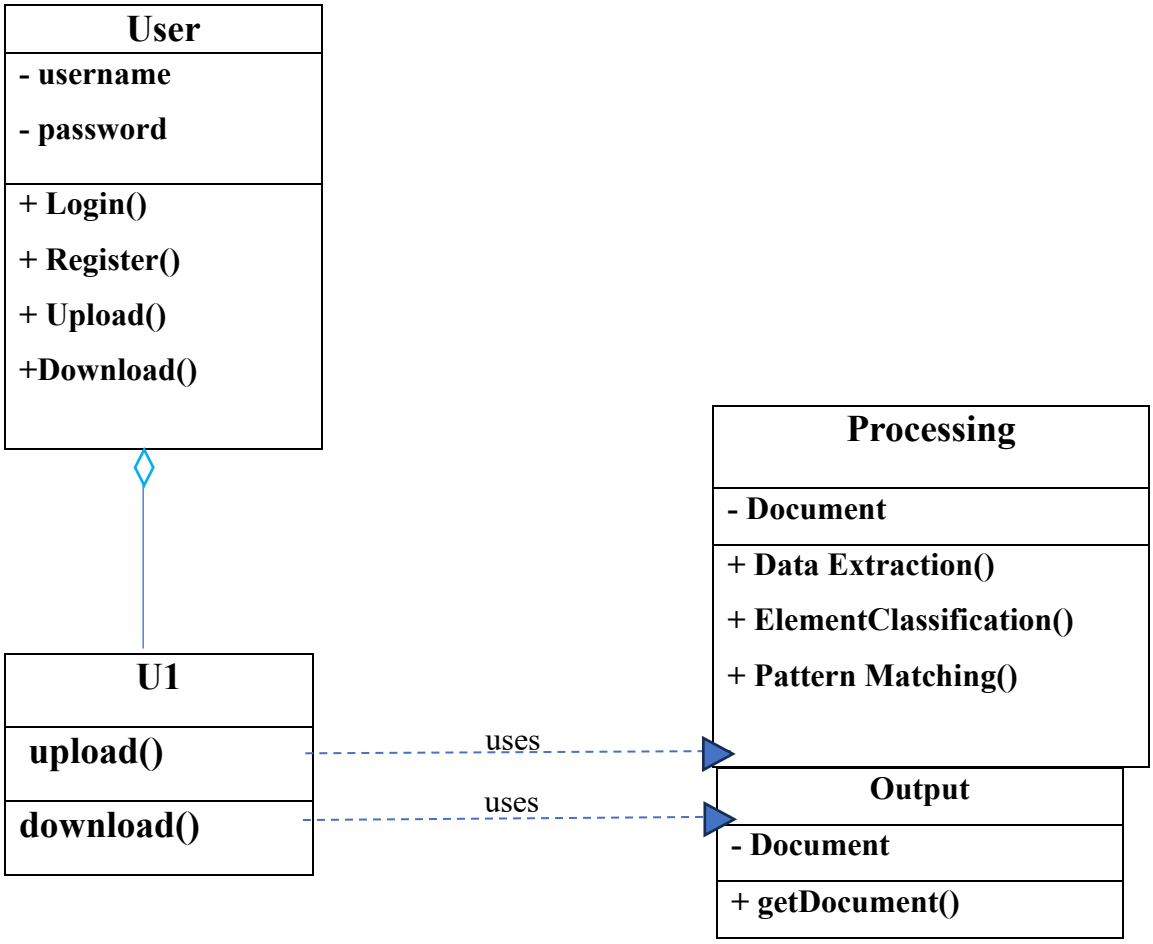

Figure 3.2 Class Diagram of Automatic word formatter

Illustrated in fig 3.2, The presented class diagram outlines three primary classes: User, Processing, and Output. In this depiction, u1 is an object instantiated from the User class, which utilizes both the Processing and Output modules. The User class encapsulates private variables for username and password, along with methods for login, registration, document upload, and download. On the Processing side, there is a private document variable, and the class hosts methods like Data Extraction, Elements Classification, and Pattern Matching, executed in a sequential manner. The Output class facilitates the user in downloading the final processed document. This design encapsulates a streamlined process where users interact by

uploading documents, triggering a sequence of processing steps, and culminating in the availability of the refined document for download.

we have three main classes: User, Processing, and Output, which together form a document processing system. The `User` class represents individuals using the system and includes functions for logging in, registering, uploading documents, and downloading processed files. Each user has a unique username and password for secure access to the system. The `Processing` class manages the actual document processing tasks. It handles tasks like extracting data from uploaded documents, classifying different document elements (like headings, paragraphs, tables, and images), and identifying specific patterns within the content. This class performs these operations sequentially to prepare the document for the next stage. Finally, the `Output` class is responsible for generating the final processed document that users can download. It consolidates the processed content into a downloadable format, allowing users to retrieve and use the refined document as needed.

In this system, a user interacts by logging in, uploading a document, and then triggering a series of behind-the-scenes processing steps handled by the `Processing` class. Once processing is complete, the `Output` class makes the refined document available for download. This streamlined process ensures that users can easily upload documents and obtain processed versions without needing to understand the technical details of data extraction, element classification, or pattern matching. The system simplifies document management by automating these tasks, making it convenient for users to work with their documents efficiently.

3.6.2 Sequence Diagram

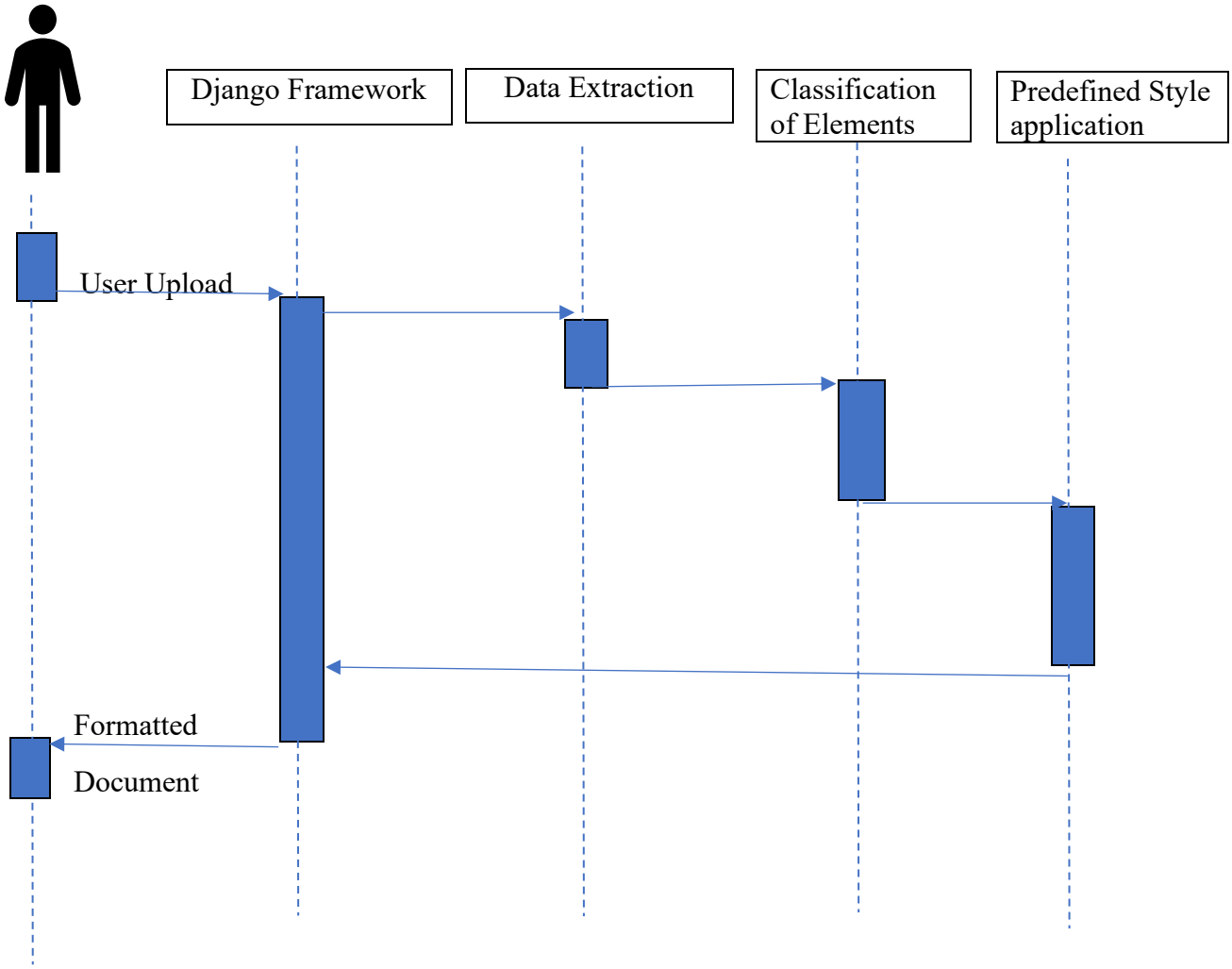

Figure 3.3 Sequence Diagram of Automatic word formatter

Illustrated in fig 3.3, The sequence begins with the "User (upload)" action, where a document is securely uploaded through Django Framework into the MongoDB database. Following this, the "Data Extraction" step utilizes the Python Docx module to extract textual and structural components from the uploaded document. Subsequently, the Classification of Elements employs a machine learning algorithm to categorize elements, setting the stage for the Predefined Style Application. Here, Python's regex module applies formatting styles based on predefined patterns there by improves the overall style and formatting of the document it also sets the layout of the document. The sequence concludes with User (download), providing the user with a professionally formatted document, completing a streamlined and efficient process.

The document processing workflow begins with the "User (upload)" action, where a user securely uploads a document using the Django Framework into a MongoDB database. This step ensures the safe storage and accessibility of the document within the system. Following the upload, the "Data Extraction" phase utilizes the Python Docx module to extract both textual content and structural components from the uploaded document. This process extracts essential information while preserving the document's layout and formatting.

Subsequently, the "Classification of Elements" stage employs a machine learning algorithm to categorize different elements within the document. This classification step is crucial for identifying headings, paragraphs, tables, images, and other components, setting the groundwork for further processing. The subsequent "Predefined Style Application" step utilizes Python's regex module to apply formatting styles based on predefined patterns. This enhances the document's overall style, corrects inconsistencies, and establishes a consistent layout, ensuring a polished and professional appearance. The sequence concludes with the "User (download)" action, where the user can download the professionally formatted document. This final step completes a streamlined and efficient process, allowing users to interact with the system easily and obtain well-formatted documents without the need for technical expertise in document processing technologies. This integrated workflow exemplifies the automation and simplification of document processing tasks, enhancing user experience and productivity.

3.6.3 Use case diagram

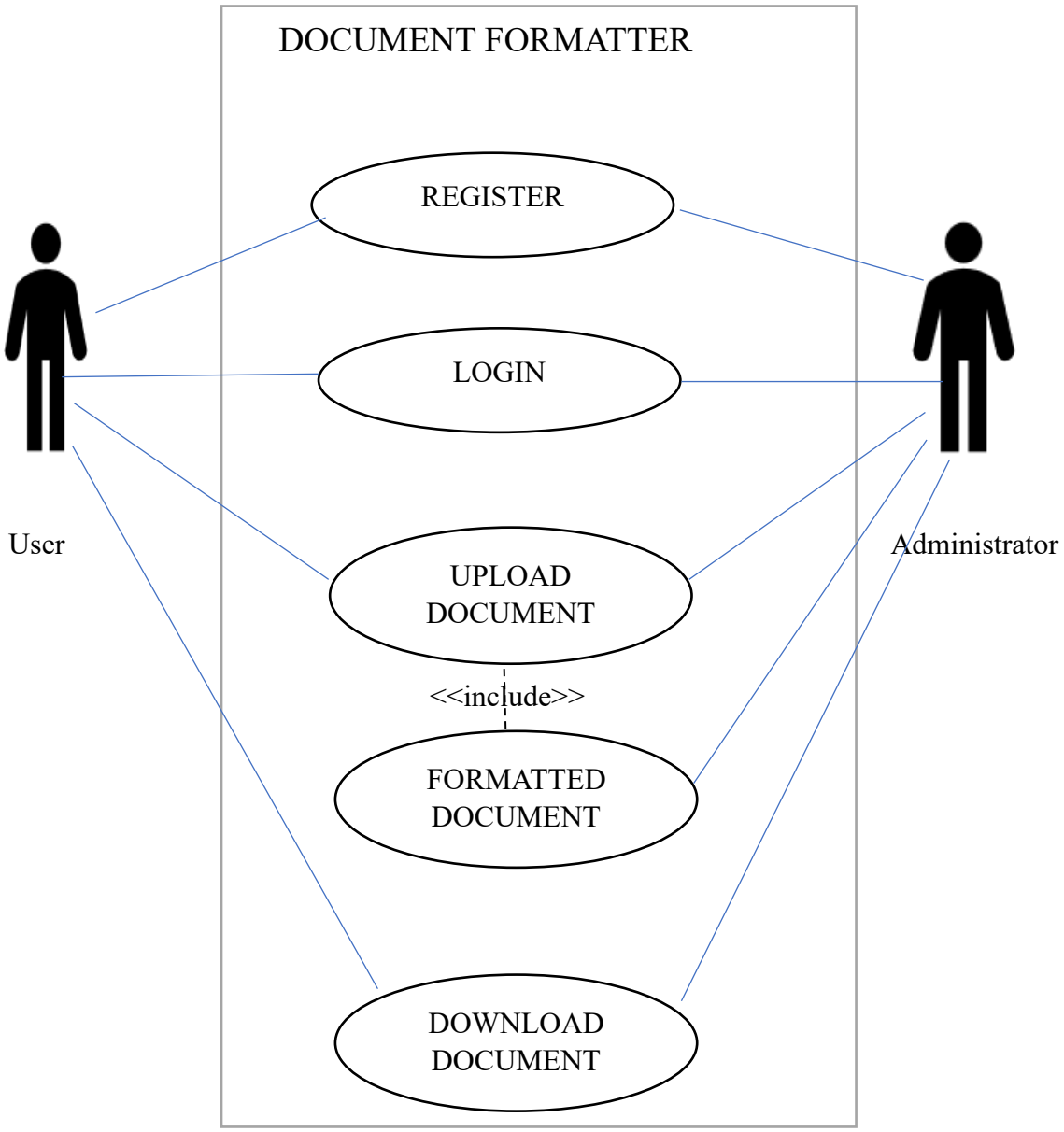

Figure 3.4 use-case diagram of Automatic word formatter

The fig 3.4 illustrates three primary functionalities: "Upload Document," where users upload Word documents; "Format Document," where the system extracts, identifies, and formats document elements, storing them in MongoDB; and "Download Document," allowing users to download the formatted document.

The following is a sequence of events that occur when a user uses the Automated Word Formatter:

The user uploads a Microsoft Word document to the platform.

When a user interacts with the Automated Word Formatter platform, they initiate the process by uploading a Microsoft Word document through the designated "Upload Document" functionality. The platform ensures compatibility by validating supported file types, typically accepting .docx format for Microsoft Word documents. During the upload, the system provides real-time feedback on the file upload progress, keeping the user informed throughout the process.

The system extracts the data from the document.

Upon successful upload, the system begins extracting data from the uploaded Word document. This extraction process involves reading and capturing text content, formatting details, and structural information embedded within the document. Additionally, metadata such as document title, author, creation date, and modification date may be extracted to enrich the document's information.

The system identifies the different elements in the document.

After data extraction, the system proceeds to identify and categorize different elements present in the document. Textual content is segmented into paragraphs, headings, subheadings, and other semantic units based on formatting and structure. Non-textual elements such as images, tables, headers, footers, and captions are also recognized and categorized for further processing.

The system formats the elements according to the predefined formatting styles.

With elements identified, the system applies predefined formatting styles or templates to each element based on its type and context within the document. Text elements are formatted for font style, size, color, alignment, indentation, and spacing according to predefined guidelines. Images and tables undergo adjustments in layout, size, and positioning to align with the document's formatting standards.

The system stores the formatted document in the MongoDB database.

Once formatting is completed, the system stores the formatted document along with its metadata and extracted elements in a MongoDB database. This storage process ensures data persistence and enables efficient retrieval of the formatted document for subsequent actions or future reference. Versioning or revision history features may also be implemented to track document changes over time.

The system generates a download link for the formatted document.

After storing the formatted document, the system generates a unique download link associated with the specific document version. This download link is automatically generated and made available to the user, providing direct access to the formatted document from the platform interface. The download link remains valid for a specified period or indefinitely, depending on system settings.

The system sends the download link to the user.

Upon completion of the formatting and storage process, the system notifies the user via email or on-screen notification that the formatted document is ready for download. The download

link, previously generated by the system, is included in the notification, allowing the user to access and download the formatted document conveniently.

The user clicks on the download link and downloads the formatted document.

Finally, the user clicks on the provided download link to initiate the download process. The formatted document is securely downloaded to the user's device, preserving the applied formatting, structure, and content as processed by the Automated Word Formatter platform. This seamless interaction completes the workflow, empowering users to efficiently obtain and utilize the formatted document for their intended purposes.

3.6.4 Activity Diagram

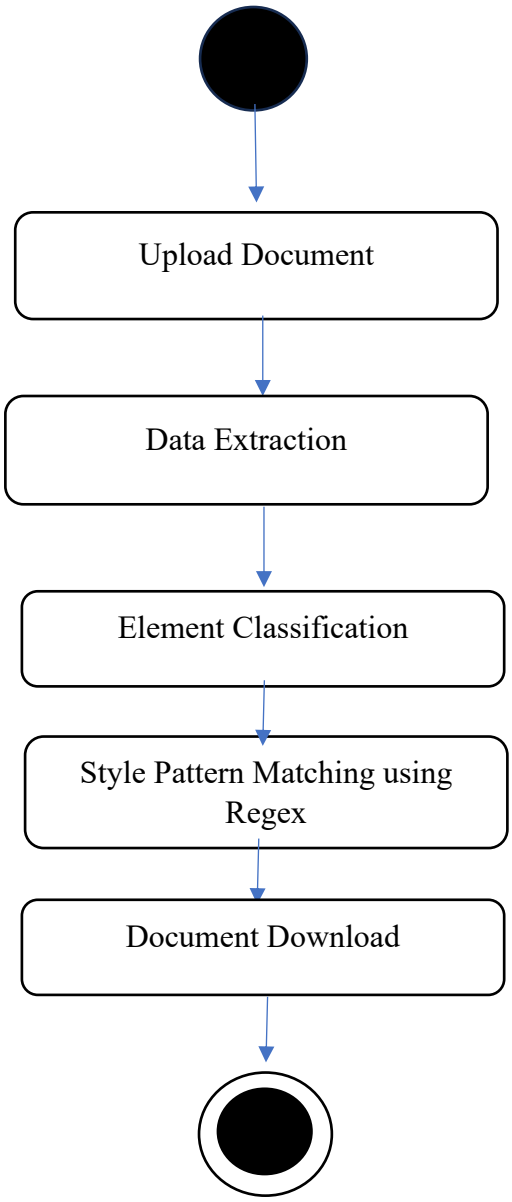

Figure 3.5 Activity Diagram of Automatic word formatter

The structured sequence of actions depicted in Figure 3.5 outlines a streamlined document processing workflow designed to automate and optimize the formatting of uploaded

documents. The process begins with the "Upload Document" activity, where users input their documents into the system through a user-friendly interface. This initial step is crucial as it initiates the entire processing pipeline, enabling users to seamlessly interact with the system and submit their documents for formatting.Subsequently, the system proceeds to the "Data Extraction" phase, where relevant data is extracted from the uploaded document. This involves parsing and capturing textual content, structural elements, and other essential information needed for further processing. The extracted data then undergoes the "Element Classification" activity, where different elements within the document are categorized. This classification step is pivotal in identifying headings, paragraphs, tables, images, and other components, setting the stage for the application of predefined formatting styles.

Moving forward, the document processing system applies "Style Pattern Matching using Regex" to automatically match and apply predefined formatting styles based on identified patterns within the document. This dynamic approach ensures consistent and accurate formatting across different document elements. Finally, the workflow concludes with the "Document Download" activity, providing users with the option to retrieve the processed and refined document in a downloadable format. This systematic and efficient progression of tasks, from document input to the application of formatting styles, underscores the system's commitment to delivering a user-centric and automated solution for document formatting, ultimately enhancing productivity and user satisfaction. By leveraging these sequential actions, the system streamlines the complex process of document formatting, allowing users to focus more on content creation and research while ensuring professional and standardized document outputs.

CHAPTER 4

Results and Discussions

4.1 Experimental Results

Figure 4.1 User Interface

The project's landing page as shown in fig 4.1, is designed with a user-centric approach, featuring three key components. Firstly, a prominent section outlines the core offerings of the platform, succinctly detailing the automated formatting service according to research paper standards. This ensures that users quickly grasp the primary functionality of the Automated Word Formatter. Secondly, a user-friendly feedback form is seamlessly integrated, allowing users to provide valuable input and suggestions. This element fosters a sense of engagement and inclusivity, encouraging users to share their experiences and contribute to the platform's continuous improvement. Lastly, the landing page incorporates a streamlined upload and download option for Word documents. This intuitive feature empowers users to effortlessly navigate the platform, upload their documents for formatting, and conveniently download the formatted results. Together, these components create a well-rounded landing page that prioritizes clarity, user interaction, and efficient document processing.

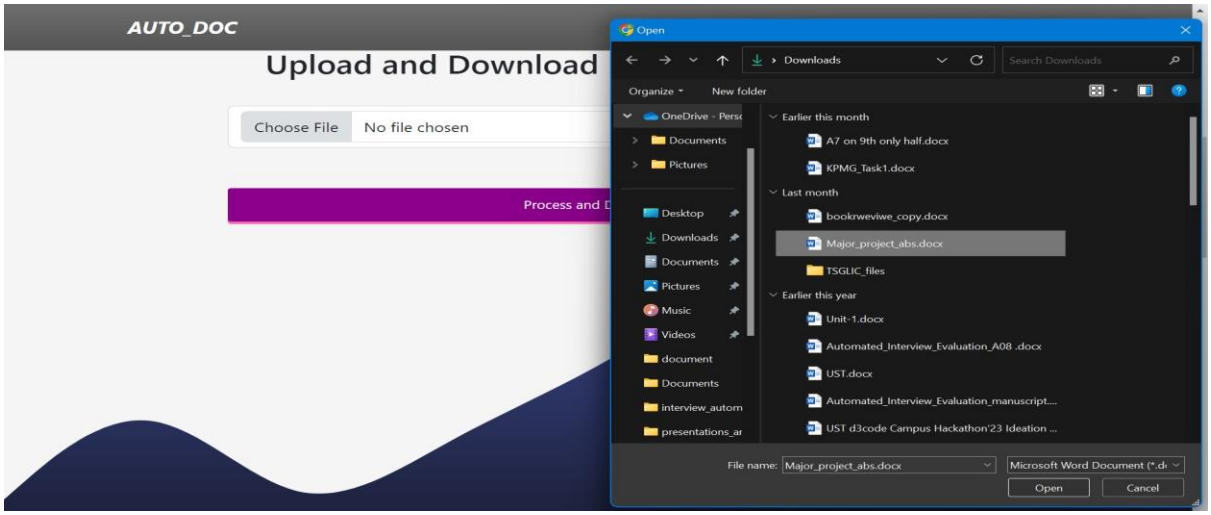

Figure 4.2 Uploading Document

Illustrated in Figure 4.2, the user engagement process starts with a clear and user-friendly interface. Users begin the document upload process by interacting with the "Choose File" button, which prompts a window to select the desired document from the file manager. After selecting the file, users have the option to reselect the file or proceed by clicking the "Process and Download" button provided on the interface. This straightforward workflow ensures ease of use and efficiency for users interacting with the system.

```
PS C:\Users\nysha\OneDrive\Desktop\major project\document> python -u "c:\Users\nysha\OneDrive\Desktop\major project\tests\new_t
ests.py"
                                      para_text table_id            style
0                     Automated Interview Evaluation  Novalue            Title
1   Ch. Sri Latha 1, N. Krishna Kalyan 2, J. Abhil...  Novalue  Author Last Name
2   1Assistant Professor, Department of AIMLE, GRI...  Novalue      Affiliation
3   2UG Student, Department of AIMLE, GRIET, Hyder...  Novalue      Affiliation
4                                                      Novalue      Affiliation
..                                                 ...      ...              ...
81  Ren, Y., & Qin, T. (2019). Fast and High-Quali...  Novalue   References Body
82  Zhong, Zhen, et al. "ALBERT: A Lite BERT for S...  Novalue   References Body
83  Zena, H., & Tokuda, K. (2012). Statistical par...  Novalue   References Body
84    “Getting Started with RNN” [Online]. Available:  Novalue   References Body
85  “https://www.pluralsight.com/guides/getting-st...  Novalue   References Body

[86 rows x 3 columns]
['Title' 'Author Last Name' 'Affiliation' 'Normal' 'Body Text'
 'Paragraph_first' 'Novalue' 'List Paragraph' 'Paragraph' 'Table content'
 'References Body']
```

Figure 4.3 Classifying of Style

Once the user uploads a document, the system performs automatic classification of document elements based on their styles, distinguishing between images, paragraphs, and tables, and identifying the specific style of each text component. For instance, Figure 4.3 illustrates how the paragraph text field "Automated Interview Evaluation" was accurately mapped to the style of "Title". This systematic approach ensures that all text contents, along

with other elements, are precisely categorized and assigned their respective styles for comprehensive analysis and processing within the document.

The style is parsed using XML, the paragraph styles undergo re-verification and validation using the XG Boost classification model. This model distinguishes paragraphs as headings or non-headings based on assessments of size, letter count, and word count within the extracted text. Achieving an impressive accuracy of 92% on a randomly generated test dataset, the XG Boost model demonstrates robust performance in accurately classifying paragraph styles, enhancing the reliability and effectiveness of the overall document analysis process.

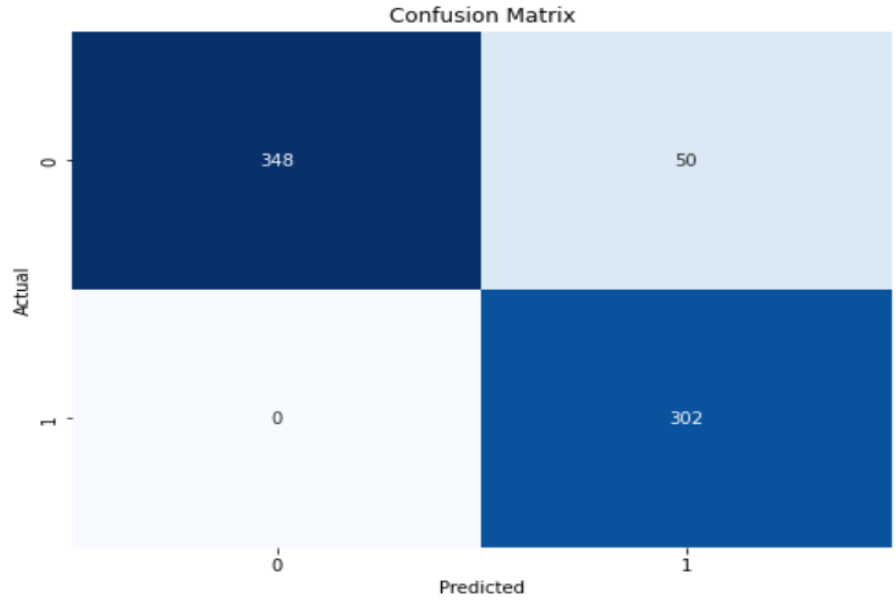

Figure 4.4 Confusion Matrix of XG Boost Classifier

Figure 4.4 presents the confusion matrix of the XG Boost classification model used to validate paragraph styles after parsing with XML. The matrix illustrates the model's performance in classifying paragraphs as headings or non-headings based on features like size, number of letters, and words. The model achieved an impressive accuracy of 92% on a randomly generated test dataset. The confusion matrix visually displays the model's ability to correctly predict headings and non-headings, showcasing the distribution of true positives, true negatives, false positives, and false negatives, providing valuable insights into the model's classification performance.

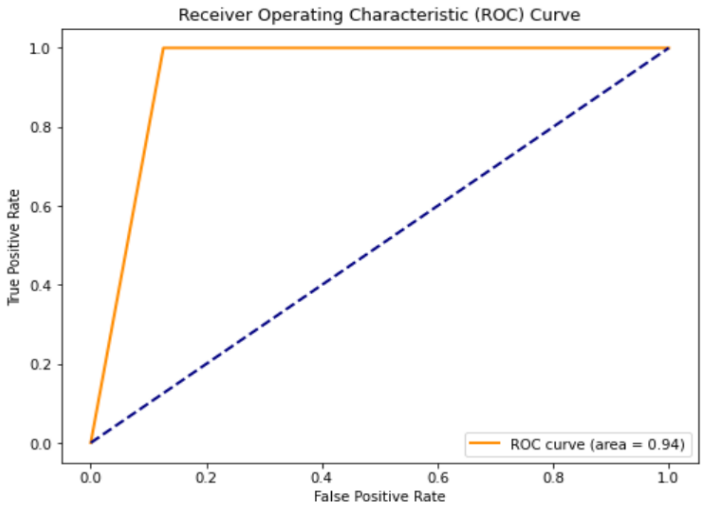

Figure 4.5 Roc Curve of XG Boost Classifier

Figure 4.5 shows the ROC curve of the XG Boost classification model used to validate paragraph styles after XML parsing. This curve demonstrates the model's effectiveness in distinguishing between headings and non-headings by plotting the true positive rate against the false positive rate across various threshold values. The ROC curve provides valuable insights into the model's ability to balance sensitivity and specificity in paragraph style classification tasks, complementing the findings from the confusion matrix Figure 4.4.

Classification Report:

	precision	recall	f1-score	support
0	1.00	0.87	0.93	398
1	0.86	1.00	0.92	302
accuracy			0.93	700
macro avg	0.93	0.94	0.93	700
weighted avg	0.94	0.93	0.93	700

Figure 4.6 Classification Report

The Figure 4.6 report shows the performance of a classification model on a binary classification task. The two classes are represented by 1 and 0. The precision, recall, and F1-score are all high for both classes, which means the model is performing well.Precision is a measure of how accurate the model is. A high precision means that most of the time the model

predicts class 1, it is actually class 1. In this case, the precision for class 1 is 1.00 and for class 0 it is 0.86.

Recall is a measure of how well the model finds all the actual positives. A high recall means that the model is not missing many actual class 1 examples. In this case, the recall for class 1 is 0.87 and for class 0 it is 1.00.F1-score is a harmonic mean of precision and recall. A high F1-score means that the model is balanced between precision and recall. In this case, the F1-score for class 1 is 0.93 and for class 0 it is 0.92. The accuracy of the model is also high, at 93%. This means that the model correctly classified 93% of the examples in the test set.

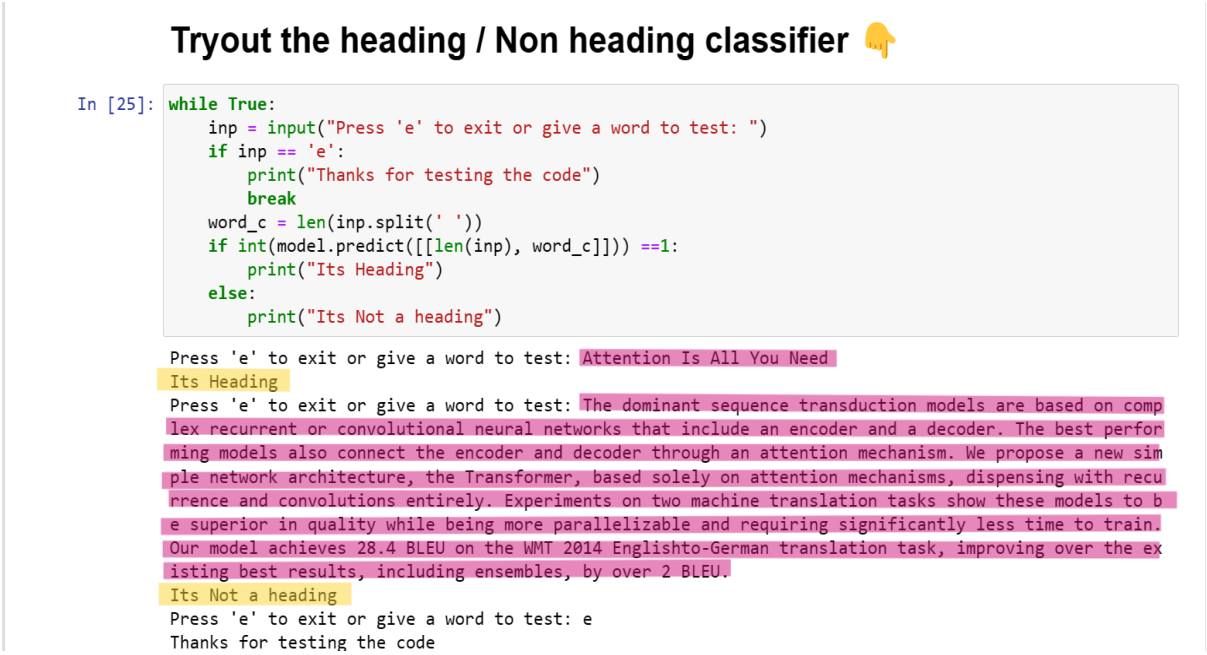

Figure 4.7 Real time results of XG Boost Classifier

As depicted in the figure 4.7 the model's purpose is to automatically classify pieces of text as either Heading or Not a Heading. The real-time aspect signifies that you can input text directly into the system, and the model will instantly analyze it and provide a classification. This allows for quick and efficient identification of headings within a document or stream of text data. In this instance, the XGBoost model has been trained on a large dataset of text examples that were already categorized as headings or non-headings. Through this training, the model learns the characteristics that distinguish headings from regular text, such as font size, position within the document, capitalization patterns, and specific keywords.

```
PS C:\Users\nysha\OneDrive\Desktop\major project> python -u "c:\Users\nysha\OneDrive\Desktop\major project\tests\new_tests.py"
S.NO   TITLE   MODEL   ADVANTAGES

1   Transformer architecture built using self and Multi-Head Attention.   Encoder and Decoder Stacks   Transform Architecture is much faster method than
Recurrent Neural Network (RNNs).

2   Transformer architecture built using encoders and decoders with dynamic halting.   Dynamic halting   Dynamic leads to faster training and better perf
ormance.

3   Bidirectional Encoder Representations from Transformers   Bidirectional Encoder Representations from Transformers (BERT).   The BERT model is built u
sing diverse datasets such as Stanford Question Answering dataset, SWAG and Winograd NLI dataset (WNLI)

ression objective function.

5.   Automatic Speech Recognition using Semi-Supervised Learning   semi-supervised learning based automatic speech recognition model (ASR)   The use of a
 conformer Encoder and LSTM Decoder architecture can improve the accuracy of the model.
```

Figure 4.8 Dealing with Tables

As shown in Figure 4.8, the process of table extraction involves converting tables into a structured data frame format followed by meticulous formatting based on the style guidelines specified in the research paper. During this formatting phase, careful attention is given to justify each row and column appropriately, ensuring a uniform and visually appealing layout. Specific cell spacing adjustments are also implemented to optimize the table's readability and comprehensibility, facilitating easy interpretation of the data presented within the table for readers and researchers alike. This methodical approach contributes to the overall clarity and effectiveness of the tables within the document analysis workflow.

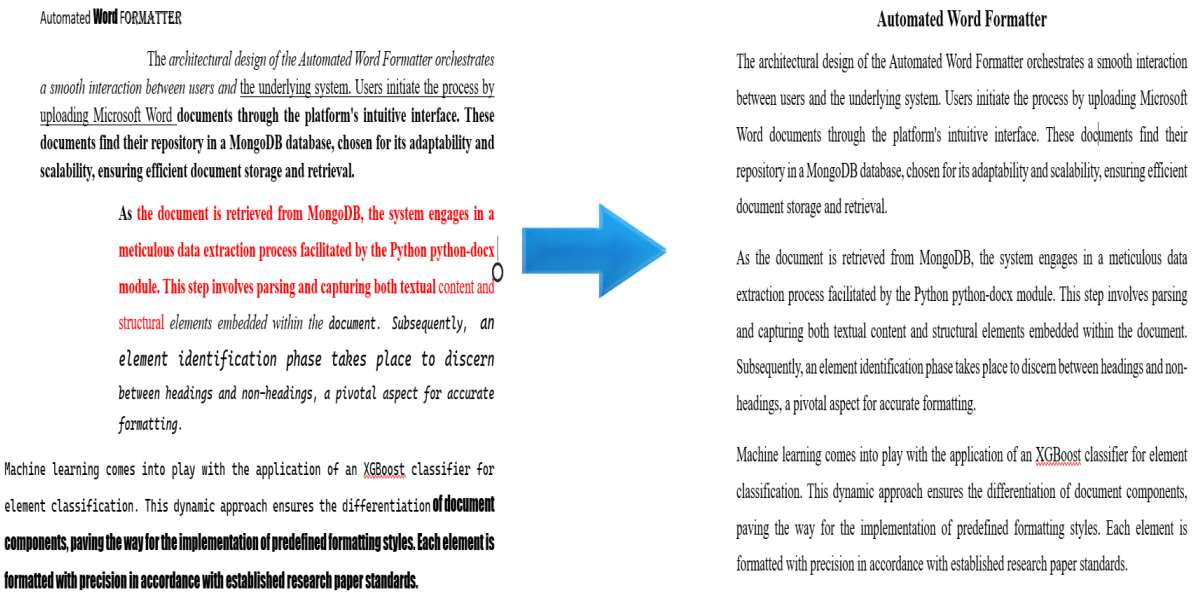

Figure 4.9 Formatted Document

Upon successful document upload, the formatting process commences, as visually represented in Figure 4.7. This pivotal stage involves a systematic approach to document enhancement. The system initiates by collecting essential data from the uploaded document, delving into its textual content and structural elements. Subsequently, an intricate element identification process takes place, distinguishing between headings and non-headings. This critical step lays the foundation for the application of predefined styles tailored to meet the specific requirements of the user. The user-centric approach ensures that the formatting aligns seamlessly with their preferences and conforms to established standards, contributing to a refined and polished document. This structured workflow underscores the system's commitment to delivering accurate and customized formatting, enhancing the overall user experience.

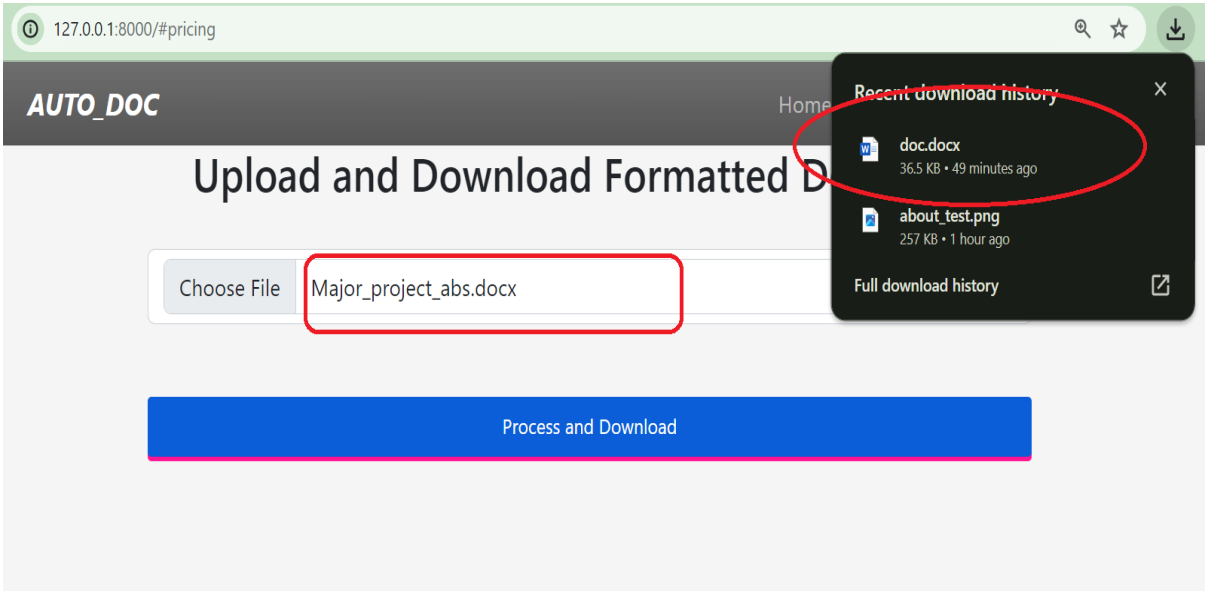

Figure 4.10 Downloading the Document

As depicted in fig 4.10, once the user uploads the required document the name of the uploaded document is displayed to the user, he can change the document if it is not the intended one after this step once the user clicks on the process and download button the document formatting process starts in the backend once the formatting is done the document gets automatically downloaded and is displayed to the user in the downloads section.

4.2 Discussion

The proposed method for automated word formatting holds significant advantages in streamlining the process of document formatting, particularly in the context of research papers. One of its key benefits lies in the efficiency and time-saving it brings to the table. By automating the formatting process, the method eliminates the need for manual adjustments,

allowing authors to focus more on content creation rather than intricate formatting details.The automated approach ensures a high level of consistency in formatting throughout the document. This is crucial for maintaining a professional and polished appearance, a challenging feat when relying solely on manual formatting. Adherence to established research paper standards is another advantage. The method is designed to meet the specific requirements of academic or industry guidelines, enhancing the chances of acceptance and publication of research work.

The user-friendly interface of the web-based platform is a noteworthy feature. Authors can effortlessly upload their documents and receive correctly formatted outputs, making the process accessible to individuals with varying levels of technical expertise. Furthermore, the automation significantly reduces the likelihood of human errors in formatting, particularly important in documents with intricate style guidelines and specific layout requirements.

Flexibility and customization are also key strengths of the proposed method. While automating standard formatting, users may have the flexibility to incorporate their preferred styles or make adjustments as needed. This strikes a balance between automation and user control. The scalability of the automated method is crucial for handling a large volume of documents efficiently, making it suitable for academic institutions, publishers, or any entity dealing with numerous documents. Additionally, the proposed method may incorporate version control features, allowing users to track changes made during the formatting process. This enhances collaboration and provides a mechanism for reviewing and reverting modifications if necessary. In conclusion, the automated word formatting method offers a transformative solution, combining efficiency, consistency, adherence to standards, and user-friendly features to streamline the document preparation process for researchers, authors, and institutions alike.

CHAPTER 5

CONCLUSION AND FUTURE ENHANCEMENTS

5.1 Conclusions

The Automated Word Formatter stands as a transformative solution, rendering document formatting a hassle-free. Users experience seamless formatting magic by effortlessly uploading their documents to the platform. This marks a significant shift, making the once tedious task of formatting documents a breeze. One of the standout features is the platform's commitment to delivering an effortless formatting experience. Users can achieve a polished and professional look for their documents without the need for intricate manual adjustments. The formatter streamlines the process, enabling users to channel their focus on the document's content rather than getting entangled in formatting intricacies. Consistency in style is a paramount achievement facilitated by the Automated Word Formatter. Fonts, alignments, and spacing are standardized, ensuring a cohesive and professional appearance across documents. This level of consistency is valuable for users aiming to maintain a unified style in their written work. A notable advantage of the formatter is its capacity to save time. The manual adjustment of paragraphs and headings becomes a thing of the past. The platform's streamlined process allows users to efficiently allocate their time to more critical aspects of content creation, knowing that formatting complexities are expertly handled. The formatter's adaptability shines through, making it a universal solution for diverse document types. Whether users are crafting a research paper or report the platform readily adapts to varying needs, providing a versatile and comprehensive formatting solution.

A key achievement of the Automated Word Formatter is its ability to ensure consistency in style across documents. Standardizing fonts, alignments, and spacing contributes to a cohesive and professional appearance, which is crucial for users aiming to maintain a unified style in their written work. Moreover, the platform saves significant time by eliminating the need for manual adjustments of paragraphs and headings. This efficiency allows users to allocate their time more effectively to critical aspects of content creation, knowing that formatting complexities are expertly managed.

The user-friendly interface further enhances the formatter's accessibility. Its intuitiveness accommodates users with varying technical expertise, ensuring a straightforward process from document upload to receiving a beautifully formatted version. This accessibility

is key to fostering a positive user experience for individuals with different levels of familiarity with formatting tools. Beyond mere formatting, the Automated Word Formatter contributes significantly to the professional presentation of documents. The platform elevates the visual appeal of written work, adding a refined aesthetic touch that is particularly advantageous for those seeking a polished and sophisticated presentation. Underlying all these benefits is the platform's commitment to conserving document integrity. Leveraging the python-docx module, the formatter ensures that the core content and structural integrity of the document remain untouched. The focus is solely on enhancing the stylistic elements, providing users with an assurance of document preservation.

5.2 Future Enhancements

These advancements can be addressed ahead in future

Additional Document Formats: Extend support beyond DOCX to include other popular document formats, ensuring a broader user base. Diverse formats like PDF, ODT, and RTF, ensures compatibility with various user preferences. This broad support fosters seamless collaboration and enhances user satisfaction, ultimately expanding your platform's reach and appeal across different industries and workflows.

Collaborative Editing and support for multiple research formats: Implementing collaborative editing features alongside support for multiple research formats such as LaTeX, Markdown, BibTeX, and HTML enriches the platform's utility for researchers. This facilitates seamless collaboration among scholars and streamlines the process of creating, sharing, and publishing academic content. With these capabilities, researchers can work efficiently within familiar formats, fostering productivity and knowledge dissemination.

CHAPTER 6

APPENDICES

A sample code of Django views which handles the process, home and download modules.

```
with open(r"C:\Users\nysha\gradient_boost_model.pkl", 'rb') as f:
        loaded_model = pickle.load(f)
    def ml_check(para):
        word_c = len(para.split(' '))
        try:
            if int(loaded_model.predict([[len(para), word_c]]))==1:
                return True
            else:
                return False
        except:
            return False
    def read_docx_tables(tab_id=None, **kwargs):
        def read_docx_tab(tab, **kwargs):
            vf = io.StringIO()
            writer = csv.writer(vf)
            for row in tab.rows:
                writer.writerow(cell.text for cell in row.cells)
            vf.seek(0)
            return pd.read_csv(vf, **kwargs)
    #    doc = Document(filename)
        if tab_id is None:
            return [read_docx_tab(tab, **kwargs) for tab in document.tables]
        else:
            try:
                return read_docx_tab(document.tables[tab_id], **kwargs)
            except IndexError:
                print('Error: specified [tab_id]: {}  does not exist.'.format(tab_id))
    for block in iter_block_items(document):
        if 'text' in str(block):
            isappend = False
            runboldtext = ''
```

```
            for run in block.runs:
                if run.bold:
                    runboldtext = runboldtext + run.text
            style = str(block.style.name)
            appendtxt = str(block.text)
            appendtxt = appendtxt.replace("\n","")
            appendtxt = appendtxt.replace("\r","")
            tabid = 'Novalue'
            paragraph_split = appendtxt.lower().split()
            isappend = True
            for run in block.runs:
                xmlstr = str(run.element.xml)
                my_namespaces = dict([node for _, node in
ElementTree.iterparse(StringIO(xmlstr), events=['start-ns'])])
                root = ET.fromstring(xmlstr)
                data
                if 'pic:pic' in xmlstr:
                    xml_list.append(xmlstr)
                    for pic in root.findall('.//pic:pic', my_namespaces):
                        cNvPr_elem = pic.find("pic:nvPicPr/pic:cNvPr", my_namespaces)
                        name_attr = cNvPr_elem.get("name")
                        blip_elem = pic.find("pic:blipFill/a:blip", my_namespaces)
                        embed_attr =
blip_elem.get("{http://schemas.openxmlformats.org/officeDocument/2006/relationships}
embed")
                        isappend = True
                        appendtxt = str('Document_Imagefile/' + name_attr + '/' + embed_attr +
'/' + str(imagecounter))
                        document_part = document.part
                        image_part = document_part.related_parts[embed_attr]
                        image_base64 = base64.b64encode(image_part._blob)
                        image_base64 = image_base64.decode()
```

```
                    dftemp =
pd.DataFrame({'image_index':[imagecounter],'image_rID':[embed_attr],'image_filename'
:[name_attr],'image_base64_string':[image_base64]})
                    image_df = image_df.append(dftemp,sort=False)
                    style = 'Novalue'
                imagecounter = imagecounter + 1
        elif 'table' in str(block):
            isappend = True
            style = 'Novalue'
            appendtxt = str(block)
            tabid = i
            dfs = read_docx_tables(tab_id=i)
            dftemp = pd.DataFrame({'para_text':[appendtxt],'table_id':[i],'style':[style]})
            table_mod = table_mod.append(dftemp,sort=False)
            table_list.append(dfs)
            i=i+1
        if isappend:
                dftemp =
pd.DataFrame({'para_text':[appendtxt],'table_id':[tabid],'style':[style]})
                combined_df=combined_df.append(dftemp,sort=False)
    combined_df = combined_df.reset_index(drop=True)
    image_df = image_df.reset_index(drop=True)
    def check(text):
        l = list(text.split(" "))
        if len(l)>9:
            return False
        return True
    target_doc = Document()
    section = target_doc.sections[0]
    section.start_type
    section.page_width = Mm(170)
    section.page_height = Mm(250)
    section.left_margin = Mm(20)
    section.right_margin = Mm(20)
```

```
        section.top_margin = Mm(24)
        section.bottom_margin = Mm(16)
        image_idx = 0
        rows = combined_df.shape[0]
        for row in range(rows):
            # Identifying a paragraph to the document
            if combined_df.iloc[row]["style"] != "Novalue":
                # target_doc.add_paragraph(combined_df.iloc[row]["para_text"])
                if  combined_df.iloc[row]["style"] == "Title":
                    para =
target_doc.add_paragraph((combined_df.iloc[row]["para_text"]).title())
                    para.paragraph_format.space_before = Mm(22)
                    para.paragraph_format.space_after = Mm(6)
                    for run in para.runs:
                        run.font.name = 'Arial'
                        run.font.size = Pt(16)
                        run.bold = True

                elif  combined_df.iloc[row]["style"] == "Section":
                    para = target_doc.add_paragraph(combined_df.iloc[row]["para_text"])
                    for run in para.runs:
                        run.font.name = 'Arial'
                        run.font.size = Pt(12)
                        run.bold = True
                elif combined_df.iloc[row]["style"] == "Affiliation":
                    para = target_doc.add_paragraph(combined_df.iloc[row]["para_text"])
                    for run in para.runs:
                        run.font.name = 'Times New Roman'
                        run.font.size = Pt(9)
                elif combined_df.iloc[row]["style"] == "Author Last Name":
                    para = target_doc.add_paragraph(combined_df.iloc[row]["para_text"])
                    for run in para.runs:
                        run.font.name = 'Times New Roman'
                        run.font.size = Pt(9)
```

```
                run.italic = True
            elif combined_df.iloc[row]["style"] == "Style Figure Caption" or
combined_df.iloc[row]["style"] == "Table Caption" :
                para = target_doc.add_paragraph(combined_df.iloc[row]["para_text"])
                for run in para.runs:
                    run.font.name = 'Times New Roman'
                    run.font.size = Pt(9)
            elif  combined_df.iloc[row]["style"] == "Subsection" or
combined_df.iloc[row]["style"] == "Subsubsection" or combined_df.iloc[row]["style"]
== "Body Text" or (check(combined_df.iloc[row]["para_text"])==True and
ml_check(combined_df.iloc[row]["para_text"])==True)and not
combined_df.iloc[row]["style"].startswith("R"):
                para = target_doc.add_paragraph(combined_df.iloc[row]["para_text"])
                for run in para.runs:
                    run.font.name = 'Arial'
                    run.font.size = Pt(10)
                    run.bold = True
            else:
                para = target_doc.add_paragraph(combined_df.iloc[row]["para_text"])
                for run in para.runs:
                    run.font.name = 'Times New Roman'
                    run.font.size = Pt(10)

            para.alignment = WD_ALIGN_PARAGRAPH.JUSTIFY
        # image is identified
        elif combined_df.iloc[row]["table_id"] == "Novalue":
            str_img = image_df.iloc[image_idx]["image_base64_string"]
            image_idx +=1
            image_bytes = base64.b64decode(str_img)
            image_stream = BytesIO(image_bytes)
            target_doc.add_picture(image_stream, width=Inches(5.0))
    #     # Table is identified
        elif combined_df.iloc[row]["table_id"] != "Novalue":
            temp_df =  table_list[table_counter]
```

```
            table_counter+=1
            table = target_doc.add_table(rows=1, cols=len(temp_df.columns))
            hdr_cells = table.rows[0].cells
            for i, col_name in enumerate(temp_df.columns):
                hdr_cells[i].text = col_name
            for _, row in temp_df.iterrows():
                new_row = table.add_row().cells
                for i, val in enumerate(row):
                    new_row[i].text = str(val)
            for row in table.rows:
                for cell in row.cells:
                    cell.vertical_alignment = WD_ALIGN_VERTICAL.CENTER
                    for paragraph in cell.paragraphs:
                        for run in paragraph.runs:
                            run.font.name = 'Times New Roman'
                            run.font.size = Pt(10)
                        paragraph.alignment = WD_ALIGN_PARAGRAPH.JUSTIFY
            table.style = 'Table Grid'
        target_doc.save('target.docx')
        return redirect('/download')
    else:
        return HttpResponse("<h1> Some Technical issue </h1>")
def download(request):
    file_path = os.path.join(os.getcwd(),"target.docx")
    if os.path.exists(file_path):
        with open(file_path, 'rb') as file:
            response = HttpResponse(file.read(),
content_type='application/vnd.openxmlformats-
officedocument.wordprocessingml.document')
            response['Content-Disposition'] = 'inline; filename="doc.docx"'
            return response
    else:
        return render(request, 'error_page.html', {'error_message': 'File not found'})
```

REFERENCES

[1] Kirill Chuvilin, "Machine Learning Approach to Automated Correction of Latex Documents," in *CONFERENCE OF FRUCT ASSOCIATION*, 2016.

[2] Oliveira, "Two Algorithms for Automatic Document Page Layout," in *Proceeding of the Eighth ACM Symposium*, 2008.

[3] Nathan Hurst, "Review of Automatic Document Formatting," in *Proceedings of the 9th ACM Symposium*, 2009.

[4] Ibrahim Nasser, "Web Application for Generating a Standard Coordinated Documentation for CS Students' Graduation Project in Gaza Universities," *International Journal of Engineering and Information Systems (IJEAIS),* vol. 1, no. 6, 2017.

[5] Graeme Gange, "Optimal Automatic Table Layout," in *Proceedings of the 11th ACM Symposium*, 2011.

[6] Mihai Bilauca, "Automatic Minimal-Height Table Layout," *INFORMS Journal on Computing,* vol. 27, 2015.

[7] John Bateman, "Towards Constructive Text, Diagram, and Layout Generation for Information Presentation," *Computational Linguistics,* vol. 27, 2001.

[8] Igor A. Bolshakov, "Text Segmentation into Paragraphs Based on Local Text Cohesion," *Springer-Verlag,* 2001.

[9] Widiastuti, "Document Image Extraction System Design," in *3rd International Conference on Informatics, Engineering, Science, and Technology (INCITEST 2020)*, 2020.

[10] "TableNet: Deep Learning Model for End-to-end Table Detection and Tabular Data Extraction from Scanned Document Images," in *International Conference on Document Analysis and Recognition (ICDAR)*, 2020.

[11] Maud Ehrmann, "Named Entity Recognition and Classification in Historical Documents: A Survey," *ACM Computing Surveys,* vol. 56, no. 2, 2023.

[12] Stefan Schweter, "FLERT: Document-Level Features for Named Entity Recognition," in *Computation and Language*, 2021.

[13] Yiheng Xu, "LayoutLM: Pre-training of Text and Layout for Document Image Understanding," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020.

[14] N. Nasyrov, "Automated formatting verification technique of paperwork based on the gradient boosting on decision trees," in *Procedia Computer Science*, 2020.

[15] Dr. Venkatesan, "An Implementation Approach Towards The Automation Of Document Formatting Using Python," *International Journal of Aquatic Science ,* vol. 12, no. 2, 2021.

[16] N. Firoozeh, A. Nazarenko, F. Alizon, and B. Daille, "Keyword extraction: Issues and methods," *Natural Language Engineering*, vol. 26, no. 3, pp. 259-291, 2020, doi: 10.1017/S1351324919000457.

[17] T. Kashinath, T. Jain, Y. Agrawal, T. Anand, and S. Singh, "End-to-end table structure recognition and extraction in heterogeneous documents," *Applied Soft Computing*, vol. 123, pp. 108942, 2022. [Online]. Available: https://doi.org/10.1016/j.asoc.2022.108942.

[18] M. Li, Y. Xu, L. Cui, S. Huang, F. Wei, Z. Li, and M. Zhou, "DocBank: A Benchmark Dataset for Document Layout Analysis," arXiv:2006.01038 [cs.CL], Jun. 2020. [Online]. Available: https://doi.org/10.48550/arXiv.2006.01038.

[19] S. Klampfl, M. Granitzer, K. Jack, and R. Kern, "Unsupervised document structure analysis of digital scientific articles," *International Journal on Digital Libraries*, vol. 14, no. 3, pp. 83-99, Aug. 2014. [Online]. Available: https://doi.org/10.1007/s00799-014-0115-1.

[20] R. Mohemad, "Automatic Document Structure Analysis of Structured PDF Files," *International Journal of New Computer Architectures and their Applications (IJNCAA)*, vol. 2, pp. 404-411, Jan. 2011.

[21] V. Naik, P. Patel, and R. Kannan, "Legal Entity Extraction: An Experimental Study of NER Approach for Legal Documents," International Journal of Advanced Computer Science and Applications, vol. 14, no. 3, 2023, pp. 775, doi: 10.14569/IJACSA.2023.0140389.

[22] Joshua C. Denny, Anderson Spickard, Kevin B. Johnson, Neeraja B. Peterson, Josh F. Peterson, Randolph A. Miller, Evaluation of a Method to Identify and Categorize Section Headers in Clinical Documents, *Journal of the American Medical Informatics Association*, Volume 16, Issue 6, November 2009, Pages 806–815, https://doi.org/10.1197/jamia.M3037

[23] Lu, W, Huang, Y, Bu, Y, & Cheng, QK. (2018). Functional structure identification of scientific documents in computer science. Scientometrics, 115(1), 463-486. https://doi.org/10.1007/s11192-018-2640-y

[24] X. Zhong, J. Tang, and A. Jimeno Yepes, "PubLayNet: Largest Dataset Ever for Document Layout Analysis," in 2019 International Conference on Document Analysis and Recognition (ICDAR), 2019, doi: 10.1109/icdar.2019.00166.

[25] J. M. Ponte and W. B. Croft, "Text segmentation by topic," in Research and Advanced Technology for Digital Libraries, C. Peters and C. Thanos, Eds. Berlin, Heidelberg: Springer, 1997, vol. 1324, pp. 1-1. doi: 10.1007/BFb0026725.