RESUME CLASSIFICATION

Presented by Yashika Upadhyay
Yash Kumar Roy
Mahesh Ushir

Project vision and mission

01. To create an intelligent, automated resume classification system that empowers businesses to optimize recruitment workflows, save time, and ensure that the best candidates are identified more quickly.

02.
- To simplify the hiring process by automating resume screening and categorization.
- To enhance recruitment efficiency through accurate and scalable classification models.

03. The system can prioritize resumes based on key qualifications, skills, or experience, ensuring that the most relevant candidates are reviewed first, leading to more efficient and targeted hiring decisions.

Project Flow

01 Data Collection and Preparation

We start by gathering resumes in different formats (PDF, Word, etc.) and extract essential features like skills, experience, education, and job titles. We then preprocess the data by cleaning, tokenizing, and converting it into a machine-learning-friendly format. Proper labeling is done to ensure accurate classification later on.

02 Exploratory Data Analysis (EDA)

We perform EDA to understand the data better, identifying patterns and uncovering issues such as missing values or outliers. Visualizing key attributes helps us gain insights into the data and informs the model-building process.

03 Model Building

In this step, we develop and train machine learning models, testing algorithms such as logistic regression, decision trees, or neural networks. We evaluate their performance using metrics like accuracy, precision, and recall to choose the best model for classifying resumes.

04 Model Deployment

Once the model is trained and validated, we deploy it into a production environment. This allows us to integrate it into an HR system or web application, enabling users to upload resumes and receive automatic classifications, making the hiring process more efficient.

Data Extraction

```
[ ] file_path1 = []
    category1  = []
    directory1 = "C:/Users/Yash/Desktop/ExcelR Project/P393/Dataset/P-344 Dataset/Resumes_Docx/Peoplesoft Resume"
    for i in os.listdir(directory1):
        if i.endswith('.docx') :
            os.path.join(directory1, i)
            file_path1.append((textract.process(os.path.join(directory1, i))).decode('utf-8'))
            category1.append('PeopleSoft')
```

```
df1 = pd.DataFrame(data = file_path1 , columns = ['Resume'])
df1['Type'] = category1
df1
```

	Resume	Type
0	Anubhav Kumar Singh\t\t\n\n\n\nCore Competenci...	PeopleSoft
1	G. Ananda Rayudu \n\n\t\t https://www.li...	PeopleSoft
2	PeopleSoft Database Administrator\n\n ...	PeopleSoft
3	Classification: Internal\n\nClassification: In...	PeopleSoft

```
[ ] file_path2 = []
    category2  = []
    directory2 = "C:/Users/Yash/Desktop/ExcelR Project/P393/Dataset/P-344 Dataset/Resumes_Docx/React Developer
    for i in os.listdir(directory2):
        if i.endswith('.docx'):
            os.path.join(directory2, i)
            file_path2.append((textract.process(os.path.join(directory2, i))).decode('utf-8'))
            category2.append('React Developer')
```

```
df2 = pd.DataFrame(data = file_path2 , columns = ['Resume'])
df2['Type'] = category2
df2
```

	Resume	Type
0	Name: Ravali P \n\n\n\n ...	React Developer
1	SUSOVAN BAG \n\nSeeking a challenging po...	React Developer
2	Kanumuru Deepak Reddy\n\n\n\n\t\n\n\nCAREER ...	React Developer

1. Initialize Lists:

- We begin by creating two empty lists, `file_path1` and `category1`. `file_path1` will store the extracted text from each resume, and `category1` will store the corresponding category for classification, in this case, "PeopleSoft."

2. Set Directory Path:

- The `directory1` variable is assigned the path where the resumes are stored. In this example, resumes related to "PeopleSoft" are located in the specified folder.

3. Iterate Through Files:

- We use a `for` loop to go through each file in the directory. The condition `if i.endswith('.docx')` checks if the file is a Word document (.docx format).

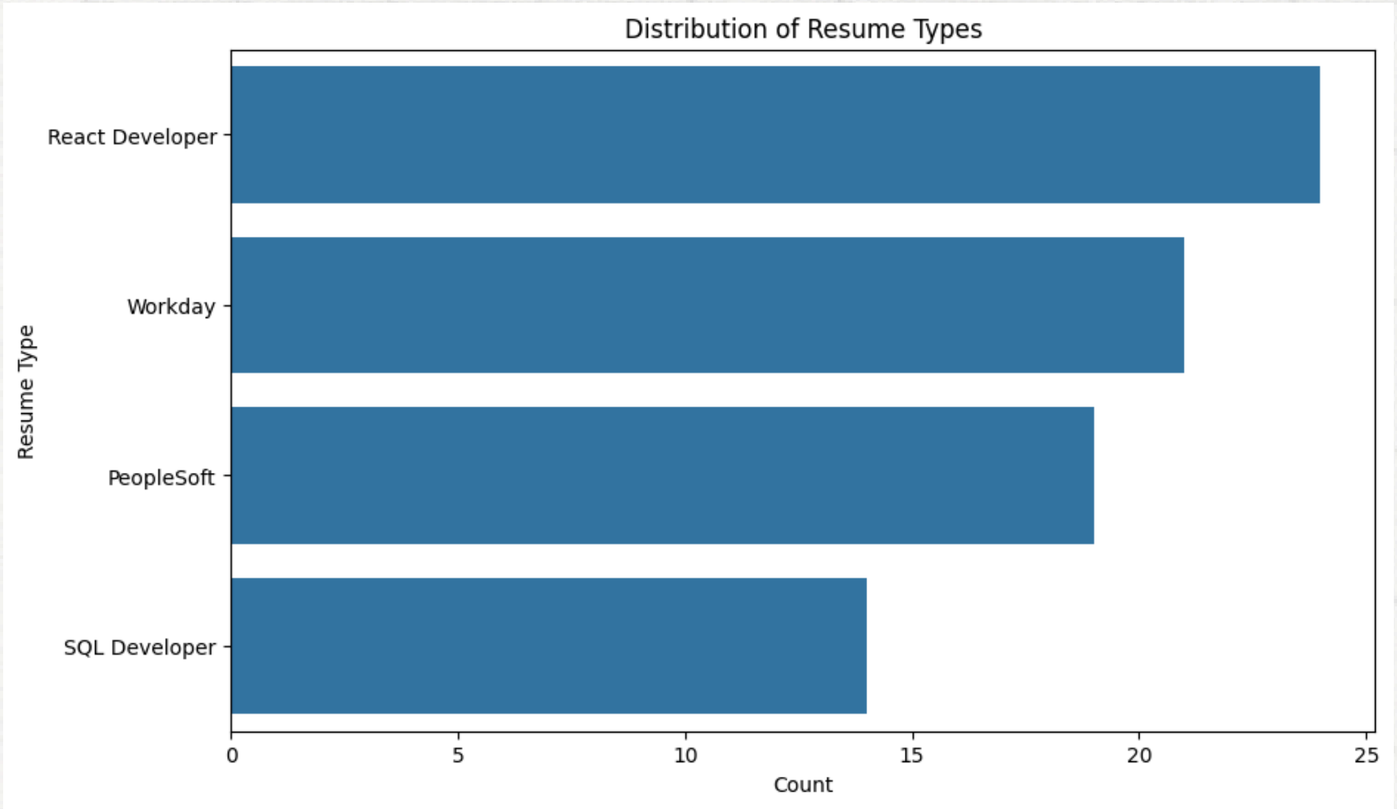
4. Extract Text from Resumes:

- For each `.docx` file, `textract.process()` is used to extract the text content. The extracted text is decoded to `utf-8` format and appended to the `file_path1` list.

5. Assign Category:

- The `category1.append('PeopleSoft')` line assigns the label "PeopleSoft" to each resume processed in this directory, helping categorize them for classification purposes.

EDA

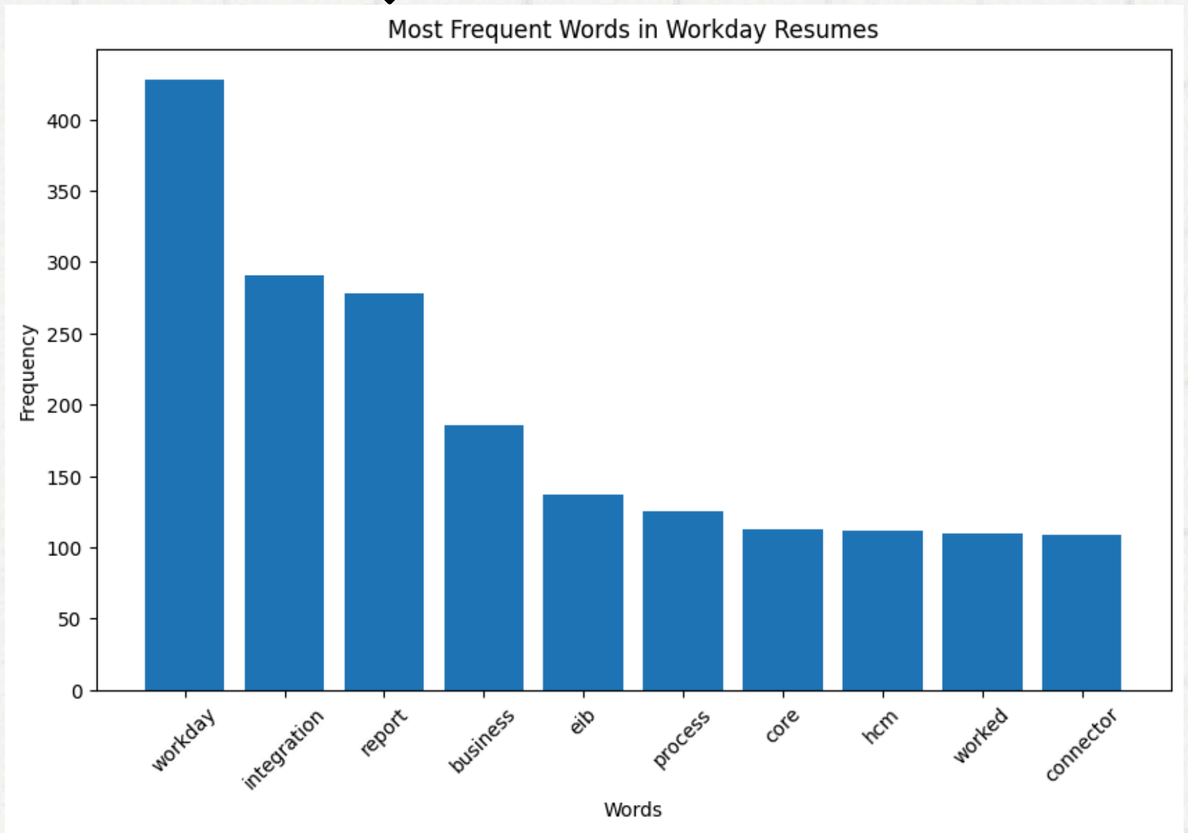


There are four types of resumes:

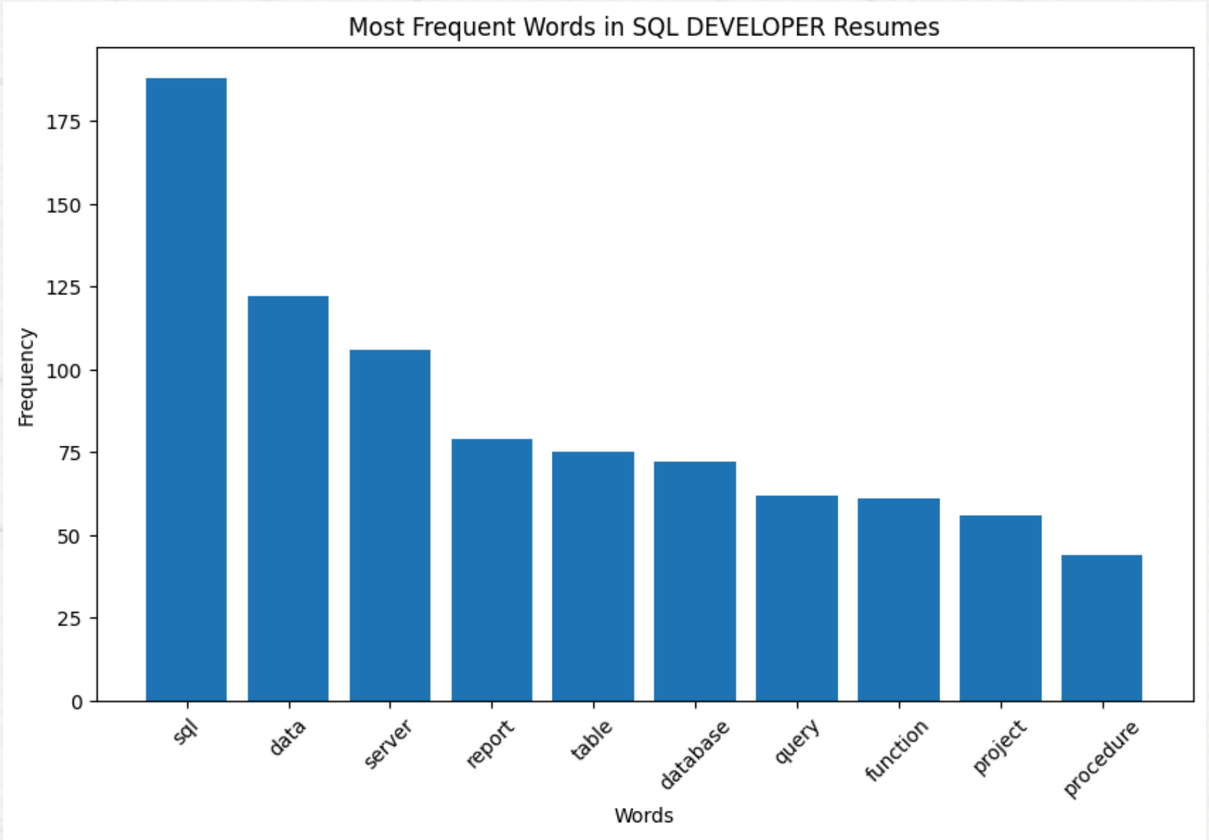
1. React Developer
2. Workday
3. PeopleSoft
4. SQL Developer

Most Common Words

WORKDAY

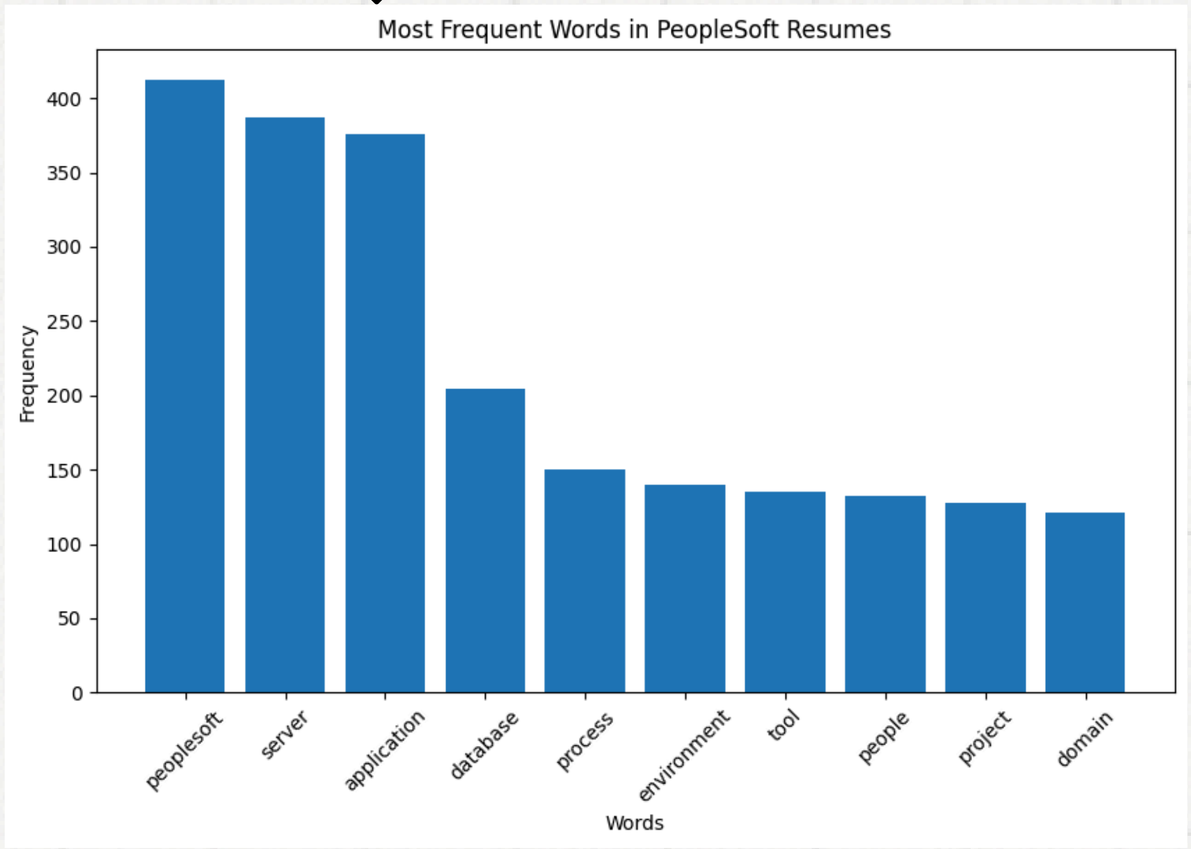


SQL DEVELOPER

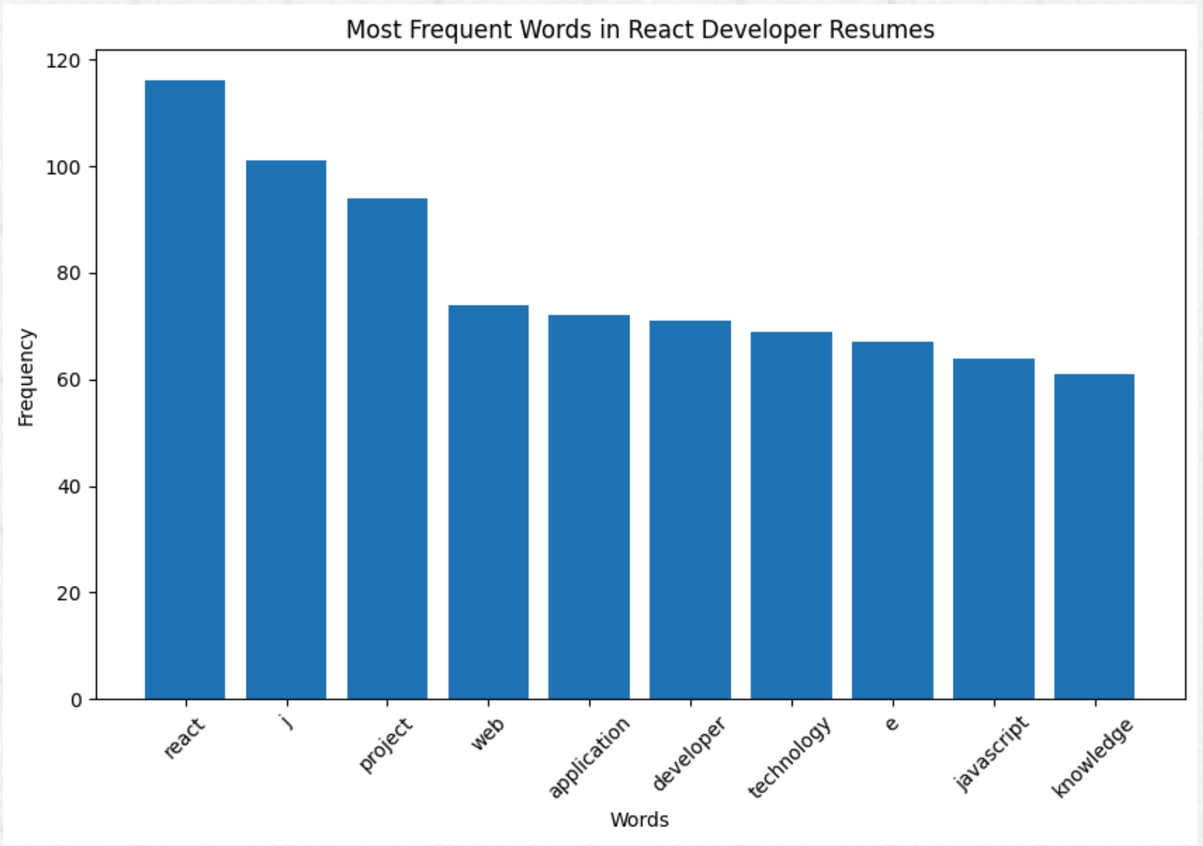


Most Common Words

PEOPLESOFT



REACT DEVELOPER



CLEANING RESUMES

REMOVING LOCATIONS

We implemented a method to remove names of states and cities from resumes. This process involves creating a list of states and cities, which is then used to generate a pattern that identifies these location names in the text. By replacing these names with blank spaces, we ensure that the resumes are devoid of geographic identifiers, further enhancing privacy and anonymization.

REMOVING PHONE NUMBER

We developed a method to remove phone numbers from resume text. This method uses a regular expression to identify 10-digit phone numbers within the text. If the text is a string, it replaces these phone numbers with empty spaces, effectively removing them. This approach is applied to the 'Cleaned_Resume' column, ensuring that personal phone numbers are removed for enhanced privacy and data protection.

REMOVING DOB

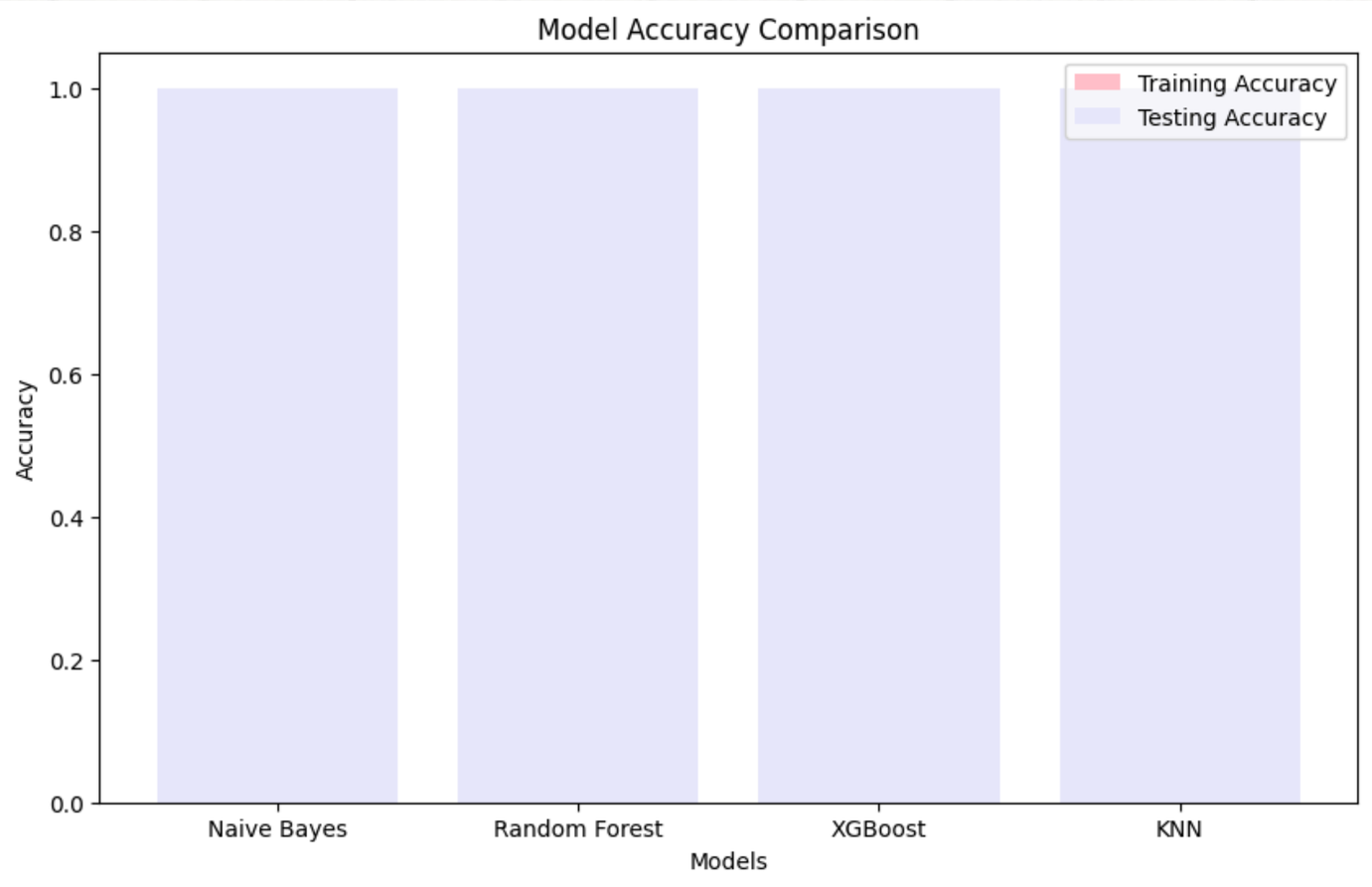
We used the re (regular expression) module to create a function that detects and removes date of birth (DOB) patterns from resumes. The remove_dob() function uses a regex pattern to identify common date formats such as DD/MM/YYYY, DD-MM-YYYY, and YYYY-MM-DD. This function is applied to the 'Resume' column, ensuring that sensitive information like DOBs is removed, keeping the data clean and privacy-compliant.

REMOVING NAME

We have implemented a method to safely remove names from resumes while preserving the content's integrity. The safe_remove_names_from_resume() function removes names from the text, but only within a limited number of words to avoid excessive removal. If too much content is removed (below a set threshold), the resume is logged as a missing resume, and the original text is retained to prevent data loss. This approach ensures that essential content is preserved while still removing sensitive information.

MODEL BUILDING

We applied the TF-IDF (Term Frequency-Inverse Document Frequency) method to preprocess our resume data. However, all models achieved 100% accuracy, indicating a clear case of overfitting. This suggests that the models performed exceptionally well on the training data but may not generalize effectively to new, unseen data. Further adjustments and validations are needed to ensure robust and reliable performance.



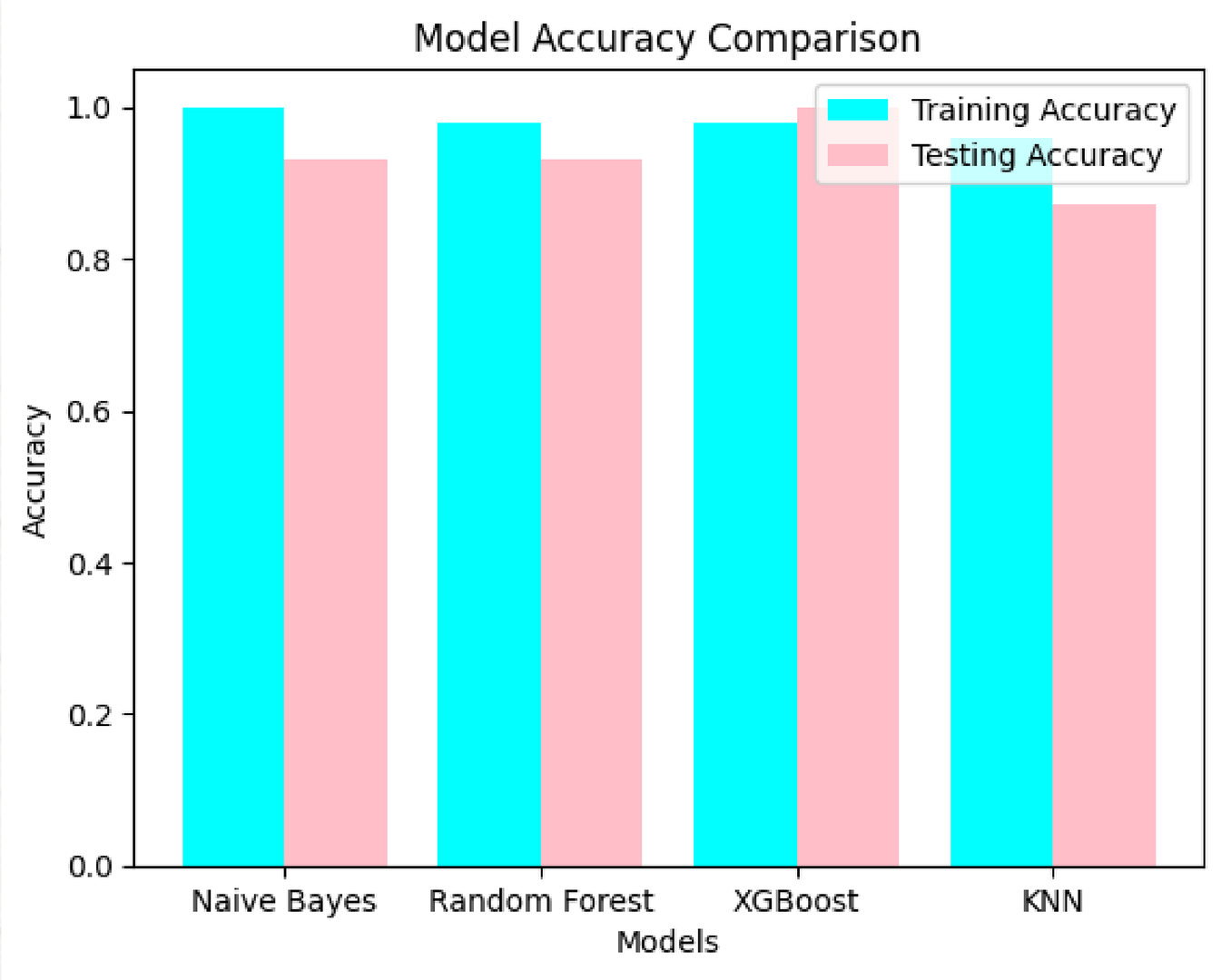
Model Performance with Count Vectorization

1. **K-Nearest Neighbors (KNN):**
- **Training Score: 0.968**
- **Testing Score: 0.875**

2. **Random Forest (Bagging):**
- **Training Score: 0.984**
- **Testing Score: 0.938**

3. **Naive Bayes:**
- **Training Score: 1.0**
- **Testing Score: 0.938**

4. **XGBoost:**
- **Training Score: 0.984**
- **Testing Score: 1.0**



1. Data Collection and Preparation:
- We gathered and cleaned resume data, removing personal information such as dates of birth, names, phone numbers, and geographic locations to ensure privacy and data anonymization.

2. Text Preprocessing:
- We applied stopword removal and used regular expressions to clean the text, removing irrelevant or common words and patterns. Lemmatization was omitted from the process to simplify the text preprocessing.

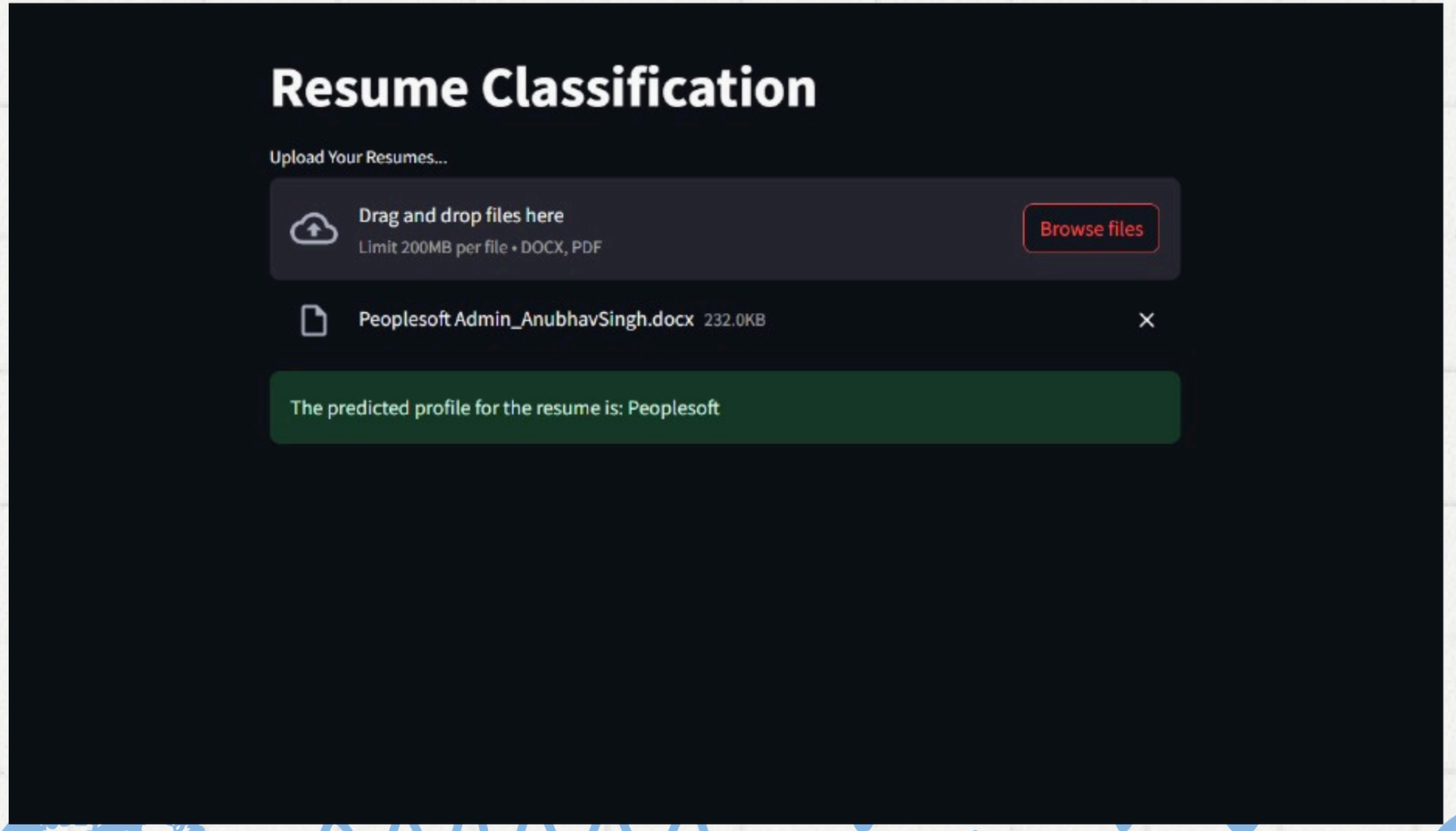
3. Feature Extraction:
- Initially, we used TF-IDF (Term Frequency-Inverse Document Frequency) for feature extraction, which led to overfitting as models showed 100% accuracy on training data but performed poorly on testing data.

4. Revised Feature Extraction:
- We switched to Count Vectorization, which resulted in a more balanced model performance.

5. Model Evaluation:
- K-Nearest Neighbors (KNN): Training Score: 0.968, Testing Score: 0.875
- Random Forest (Bagging): Training Score: 0.984, Testing Score: 0.938
- Naive Bayes: Training Score: 1.0, Testing Score: 0.938
- XGBoost: Training Score: 0.984, Testing Score: 1.0

Count Vectorization improved model robustness and reduced overfitting, with XGBoost and Naive Bayes showing strong performance on both training and testing datasets.

MODEL DEPLOYMENT



Thank you very much!