Job Scam Detection and Prevention

Sukhbir Singh Sardar, Manogna Lakkadasu, Pravalika Papasani,
Surya Theja Dokka, Maitri Hemant Mistry

INTRODUCTION

Advancement in technology has made one's life extremely facile. With these developments in technology, along with the benefits there have been many threats arising too. One of them is the Fake Job Posting, which is done by the scammers to fraud people. These scammers try to display fake job postings through which they try to get the application fee, or the personal details of the person who is trying to apply for it. We have chosen the job_fake_postings dataset from kaggle website, which was originally taken from The University of the Aegean | Laboratory of Information & Communication Systems Security, and has pretty good information through which we can build a classifier or predict results.

DESCRIPTION OF THE DATASET

Column Name	Description of Column
job_id	A unique job id
title	The title of the job and entry
location	Location of the job that posted
department	Job that belongs to which Corporate department (e.g. sales).
salary_range	The range of the salary that the job posted
company_profile	The short description the company
description	An overview of the job that posted
requirements	The list of requirements that are required for the job opening
benefits	The benefits that offered by the employer for the job that posted
telecommuting	True for telecommuting positions.
has_company_logo	This column gives the info about the company has logo

	or not
has_questions	Screening questions are present or not
employment_type	Different types of employment (eg: Full time,part time,.etc)
required_experience	This column helps about the required experience (eg: Intern,Entry level,.etc)
required_education	Level of education required (eg: Doctorate, Masters, Bachelors,..etc)
industry	Type of industry (eg: Automotive, IT, Health care, Real estate, etc.)
function	The functions that are performed by the company(eg: Consulting, Engineering, Research, Sales etc.)
fraudulent	It is the target variable used for classification

Table 1: Data set description

The dataset consists of 18k jobs and their description, out of which only 800 are fake jobs. The data includes meta-data about the occupations in addition to textual information. Using the dataset, classification algorithms that can identify fake job descriptions can be developed. It has 18 columns which describes about the data and the detail information is listed below:

EXPLORATORY DATA ANALYSIS

The dataset is imbalanced, with 95.2% of the data labeled as non-fraudulent, while only 4.8% is labeled as fraudulent. This imbalance can present challenges for machine learning models, making it biased towards the majority class. Consequently, the model may have problems correctly predicting instances of the minority class.

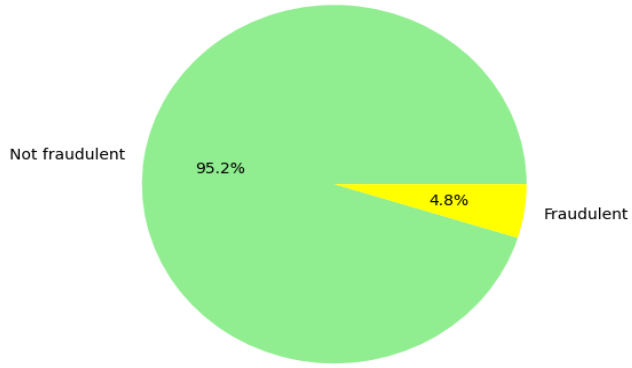

Fig 1: Pie chart of target variable

The correlation heatmap suggests that multicollinear features are not present in the dataset.

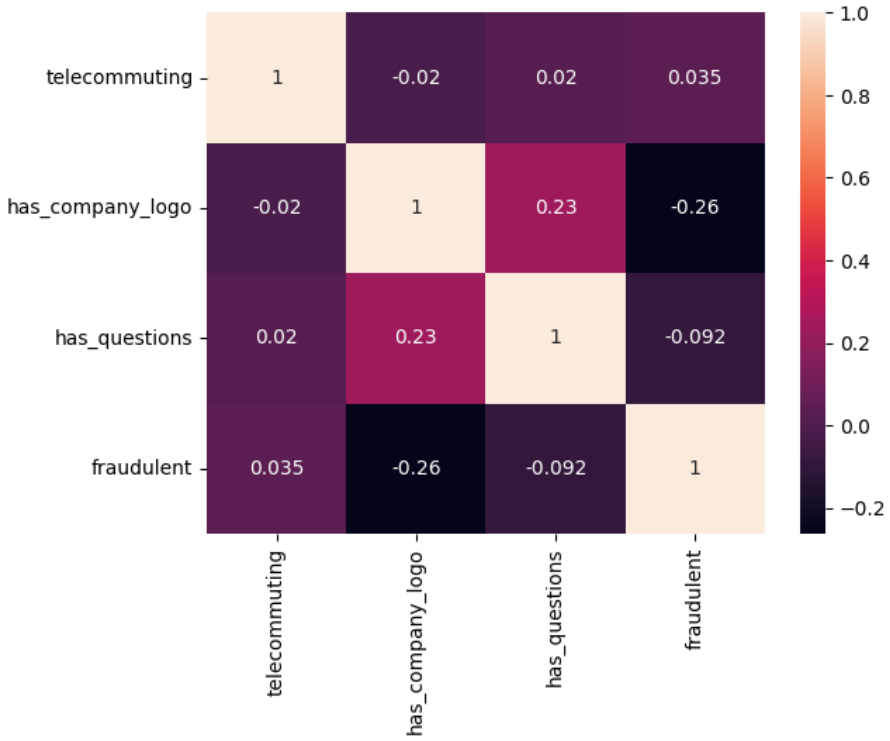

Fig 2: Correlation heatmap between attributes

We have split the dataset into fraudulent and not fraudulent dataframes in order to analyze the feature characteristics for each label. For example, we can deduce that irrespective of the labels, the majority of job postings are based in the US location.

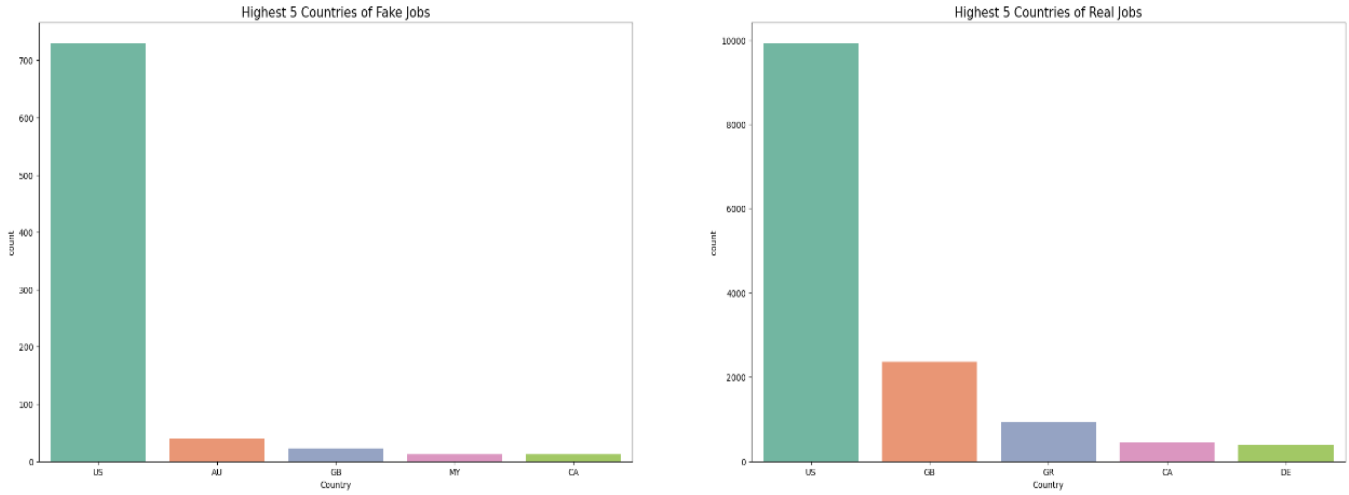

Fig 3: Bar graph that represents the highest 5 countries of both fake and real jobs

Likewise, we can infer that the majority of fake job postings are directed towards individuals seeking entry-level positions. This is logical as new job seekers may not have a full understanding of market salary norms or be as familiar with the warning signs in job postings.

Most fake job postings appear to originate from industries such as Oil & Energy, Accounting, Hospital & Healthcare, and Marketing & Advertising. Notably, the industries represented in these fake job advertisements are very different from those in actual job listings.
Administrative positions appear to be the focus of fake job advertisements.

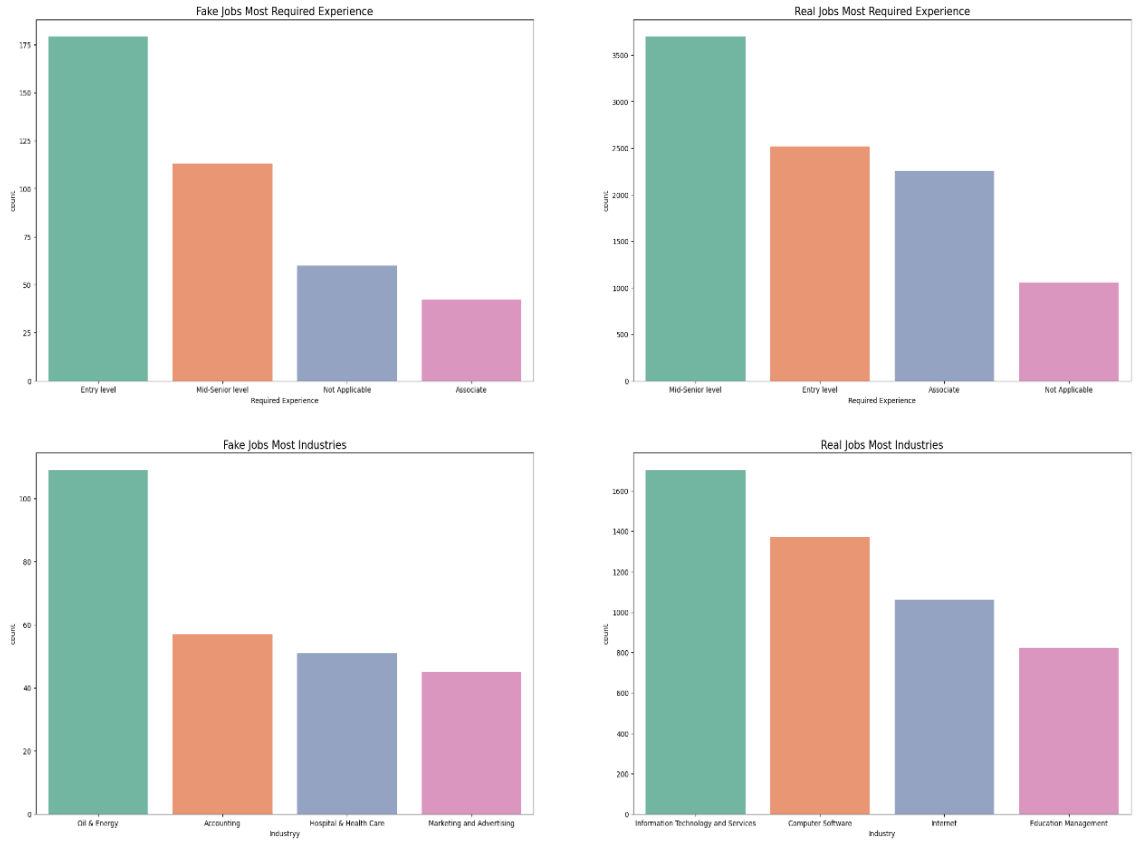

Fig 4: Bar graph that represents the most required experience and the most industries for both fake and real jobs

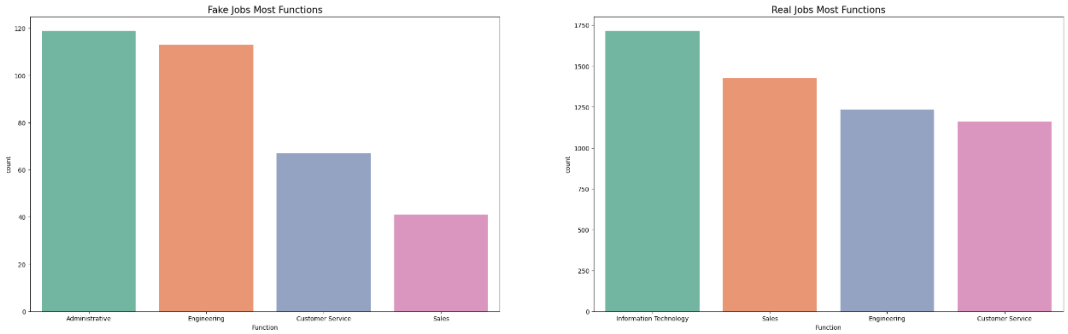

Fig 5: Bar graph that represent most common functions for fake and real jobs

We conducted a comparison of the character and word counts of descriptive columns (Description, Requirements, Benefits, and Company Profile) for both real and fake job postings to identify patterns in the fake and real posts based on the number of characters or words used in the post.
The distributions of most descriptive columns appear to be similar for both fake and real job postings. However, in the case of the 'Company Profile' column, the character count and word count tend to be higher for real job postings compared to fake ones. We came to the conclusion that using character and word counts as features might not be useful in this case.

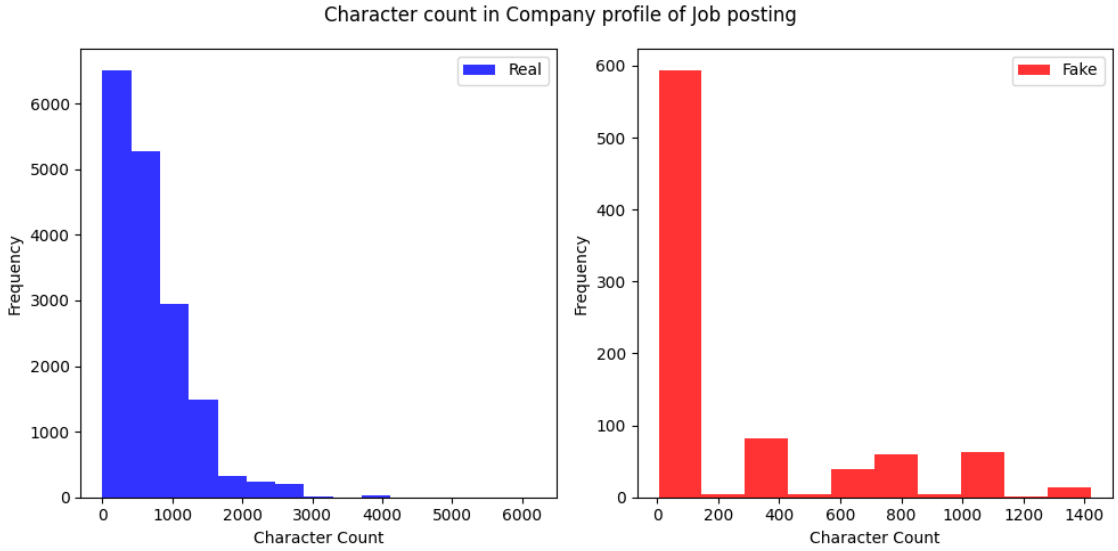

Fig 6: Character count in Company profile of job posting

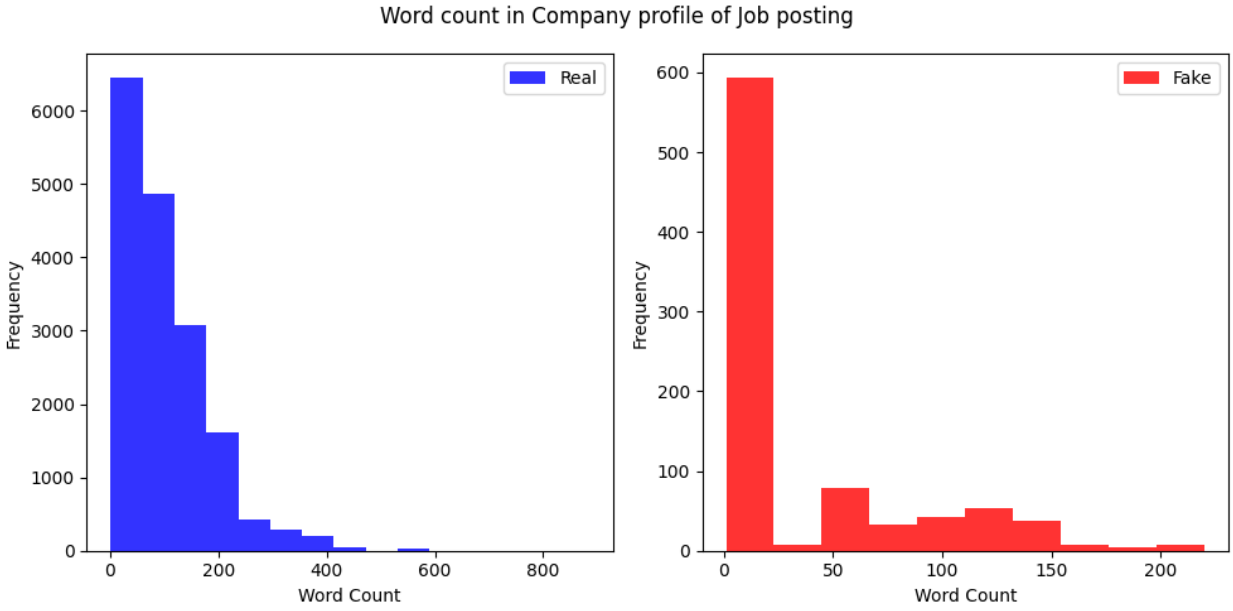

Fig 7: Word count in Company profile of job posting

Now that all the categorical and numerical values have been analyzed, it is also necessary to have a look at the text. In the job_fake_postings dataset, some of the most unique columns have a huge amount of text for each instance. They include company description, requirements, benefits etc. These columns have huge strings of data which gives important information about the job. This data is also important for contributing to identify whether the particular job is fraud or not.

Firstly all the text data was combined together so that we can work on them at a time. A list was created, where all the text data for particular instances is combined and stored in the 'text' variable. For each element in the list, the words have been tokenized(tokenize here means that they have been split into separate words). Tokenization has been done through the method word_tokenize() which takes a huge number of words as input. Once the words have been tokenized, all the letters are converted to lowercase as the machine considers upper and lower letters the same.

Some special characters such as commas(,), stops(.) etc. were also eliminated once working with lowercase letters was finished. That was done using the isalpha method, which considered all the items except the non-alphabetical characters. Then stopwords were also to be eliminated. We used the stopwords dataset to remove the unnecessary stopwords which don't contribute any information. Words such as and, a, the, which, was, were, are etc. are considered as stopwords.
There are several keywords which are in common for different types of jobs whose relevance with other keywords in their description gives a conclusion about the fraudulent data.

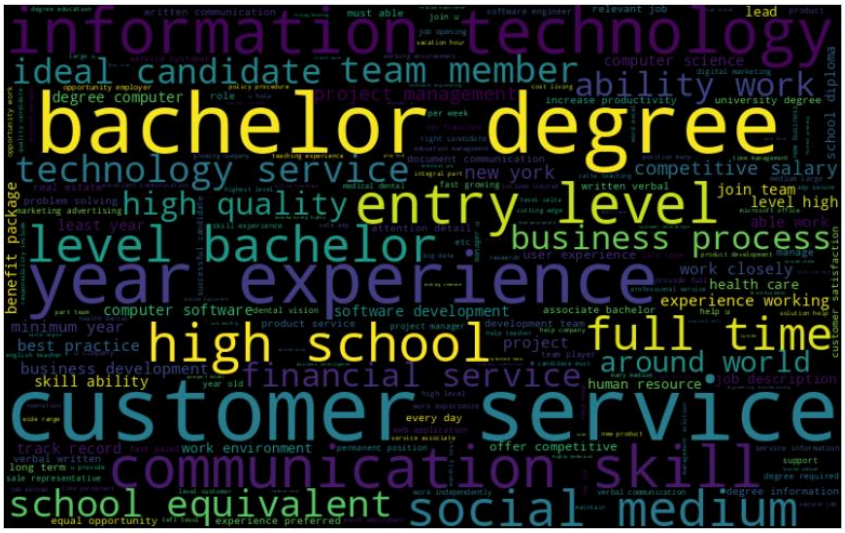

Fig 8: Word Cloud for text

In the subsequent step of text preprocessing, lemmatization was employed to further improve the tokenized words. Lemmatization involves reducing words to their base or root form, which helps in consolidating similar words and simplifying the analysis process. In this context, the NLTK library's WordNetLemmatizer was utilized to lemmatize the words in the list obtained after eliminating stopwords. Each token was processed through the lemmatizer, resulting in a set of

lemmatized words that retained the semantic meaning of the original text. For example, the word "running" would be lemmatized to "run." This step is crucial for ensuring consistency in the representation of words and enhancing the effectiveness of subsequent analysis on the dataset. Finally, A word cloud was displayed as shown in Fig 8 above.

MODELING AND RESULTS

In our study, we employed four distinct machine learning models to detect and analyze job scam postings: Logistic Regression, Naive Bayes, Random Forest, and K-Nearest Neighbors (KNN). These models were chosen for their diverse approaches to classification problems and their suitability for handling both numerical and textual data prevalent in job postings.

Each model was trained on a split of 80% training and 20% testing data. We used cross-validation to ensure the models' robustness and generalizability. The models were evaluated based on various metrics, including accuracy, precision, recall, F1-score, and ROC-AUC values.

1. Gaussian Naive Bayes (GNB)

The Naive Bayes classifier, employed in this study, is a probabilistic machine learning model known for its simplicity and efficiency, particularly in text classification tasks. It operates under the 'naive' assumption of independence between predictors, making it highly suitable for high-dimensional datasets. In the context of job scam detection, Naive Bayes efficiently handles the textual data from job postings, determining the likelihood of a posting being a scam. This model's strength in dealing with large feature spaces and its ability to rapidly make predictions make it a valuable tool in quickly filtering out potential job scams.

Performance Metrics:

- **Precision:** For legitimate jobs (class 0), the precision was 94%, but it dropped to 60% for scams (class 1). This disparity indicates a higher accuracy in identifying legitimate jobs over scams.
- **Recall:** The model showed a 34% recall for class 0 and an impressive 98% for class 1, highlighting its effectiveness in identifying most scam postings.
- **F1-Score:** The F1-scores were 50% for class 0 and 74% for class 1, reflecting a better balance in detecting scams than legitimate jobs.

Accuracy and Confusion Matrix:

- **Overall Accuracy:** The model achieved an accuracy of 66% across both classes.
- **Confusion Matrix:** It correctly identified 59 legitimate jobs and 169 scams, with 115 legitimate jobs misclassified as scams and only 4 scams misclassified as legitimate jobs.

Navie Bayes Model Evaluation

	precision	recall	f1-score	support
0	0.94	0.34	0.50	174
1	0.60	0.98	0.74	173
accuracy			0.66	347
macro avg	0.77	0.66	0.62	347
weighted avg	0.77	0.66	0.62	347

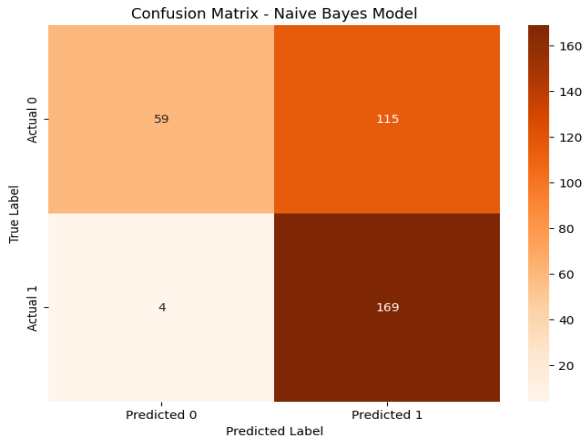

Fig 9: Classification report of Naive Bayes **Fig 10: Confusion Matrix of Naive Bayes**

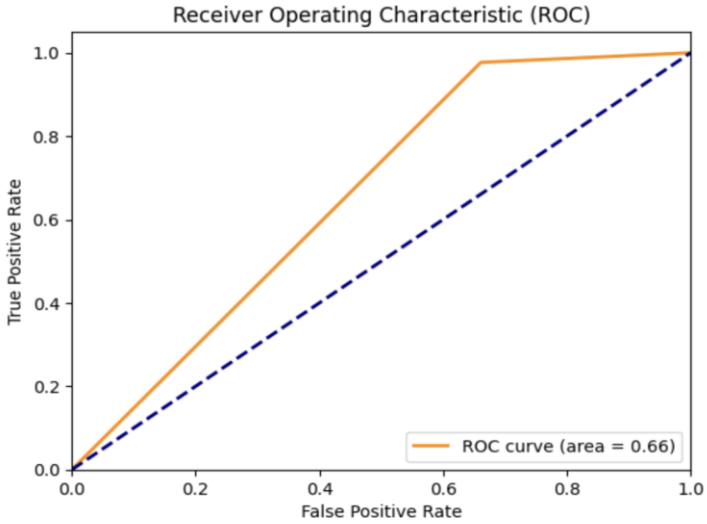

Fig 11: ROC for Naive Bayes

2. K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a simple yet effective machine learning algorithm used in our study for job scam detection. Its primary strength lies in its ability to classify data based on proximity to its nearest neighbors in the feature space. KNN makes predictions by analyzing the labels of the nearest data points, making it particularly adept at handling classification problems where the relationship between features is indicative of the outcome. This intuitive approach to classification, relying on the assumption that similar data points are close to each other, makes KNN a valuable tool for discerning between legitimate and fraudulent job postings.

Performance Metrics:

- **Precision:** For legitimate jobs (class 0), KNN achieved a precision of 80% and 75% for scams (class 1).

- **Recall:** The recall for class 0 was 74%, while it was higher for class 1 at 82%, indicating a slightly better identification of scams.
- **F1-Score:** The F1-scores were 77% for class 0 and 78% for class 1, showing a balanced performance in detecting both legitimate jobs and scams.

Accuracy and Confusion Matrix:

- **Overall Accuracy:** KNN achieved an accuracy of 78%.
- **Confusion Matrix:** It correctly identified 128 legitimate jobs and 141 scams, with 46 legitimate jobs misclassified as scams and 32 scams as legitimate jobs.

KNN Model Evaluation

	precision	recall	f1-score	support
0	0.80	0.74	0.77	174
1	0.75	0.82	0.78	173
accuracy			0.78	347
macro avg	0.78	0.78	0.77	347
weighted avg	0.78	0.78	0.77	347

Fig 12: Classification report of KNN

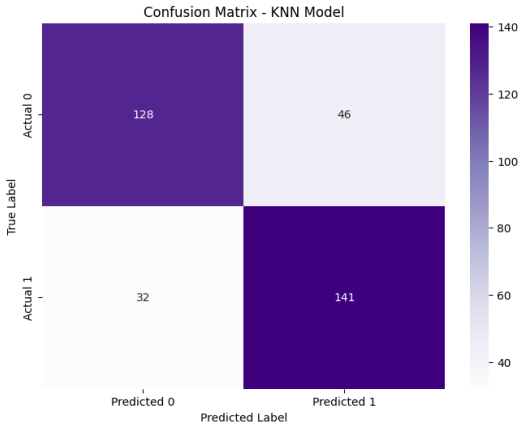

Fig 13: Confusion Matrix of KNN

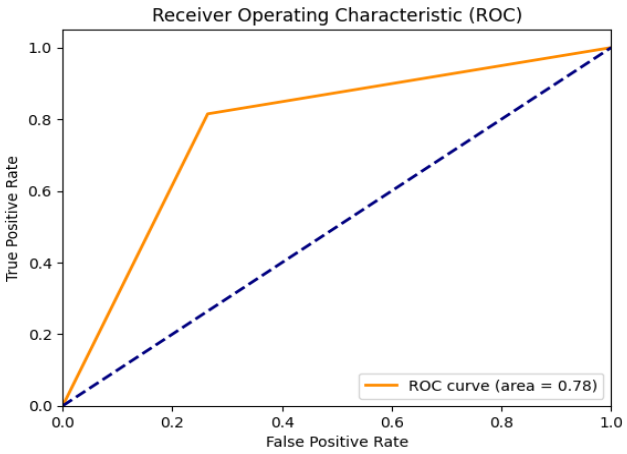

Fig 14: ROC for KNN

3. Logistic Regression

Logistic Regression, a robust binary classification algorithm, was utilized in this study for its effectiveness in distinguishing between legitimate and fraudulent job postings. Its suitability stems

from its ability to handle binary data and provide probabilistic outputs. This model is particularly beneficial in scenarios like job scam detection, where it is crucial to differentiate between two distinct categories: legitimate job listings and scams. Logistic Regression's predictive capabilities are grounded in estimating the probabilities using a logistic function, offering a balance between precision and recall, crucial for minimizing false positives in scam detection.

Performance Metrics:

- **Precision:** The model achieved 88% precision for classifying legitimate jobs (class 0) and 81% for scams (class 1).
- **Recall:** The recall was 79% for class 0 and 89% for class 1, indicating a higher sensitivity towards identifying scams.
- **F1-Score:** The F1-scores stood at 83% for class 0 and 85% for class 1, showcasing a balanced performance in detecting both categories.

Accuracy and Confusion Matrix:

- **Overall Accuracy:** Logistic Regression showed an overall accuracy of 84%.
- **Confusion Matrix:** It correctly identified 138 legitimate jobs and 154 scams, with 36 legitimate jobs misclassified as scams and 19 scams as legitimate jobs

Logistic Regression Model Evaluation

	precision	recall	f1-score	support
0	0.88	0.79	0.83	174
1	0.81	0.89	0.85	173
accuracy			0.84	347
macro avg	0.84	0.84	0.84	347
weighted avg	0.84	0.84	0.84	347

Fig 15: Classification report of Logistic regression

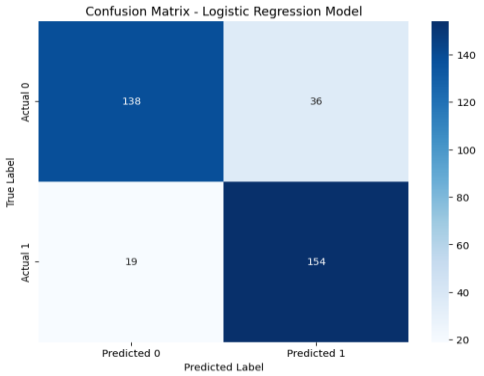

Fig 16: Confusion Matrix of Logistic regression

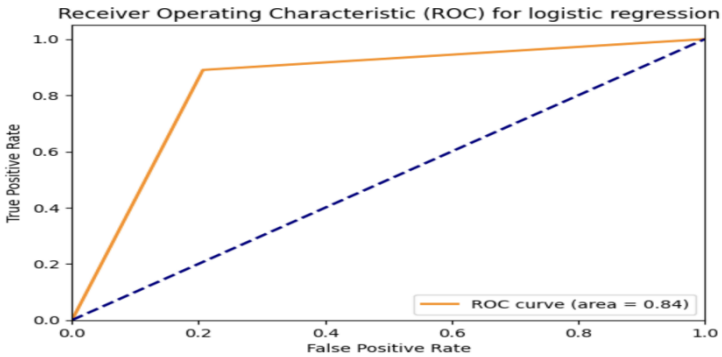

Fig 17: ROC for Logistic regression

4. Random Forest

The Random Forest algorithm, utilized in our job scam detection study, is a powerful ensemble learning method. It operates by constructing a multitude of decision trees during training and outputs the class that is the mode of the classes of individual trees. This approach makes it highly effective for classification tasks, as it can handle a large number of features and maintain accuracy even with missing data. Random Forest is particularly adept at reducing overfitting, a common challenge in complex models, by averaging multiple deep decision trees, each trained on a different part of the dataset. This robustness makes it an excellent choice for discerning fraudulent job postings from legitimate ones.

Performance Metrics:

- **Precision:** Achieved 89% for legitimate jobs (class 0) and 82% for scams (class 1).
- **Recall:** Demonstrated 80% recall for class 0 and 90% for class 1, indicating strong scam identification capabilities.
- **F1-Score:** Recorded 84% for class 0 and 86% for class 1, showing a balanced performance.

Accuracy and Confusion Matrix:

- **Overall Accuracy:** Attained an accuracy of 85%.
- **Confusion Matrix:** Correctly identified 140 legitimate jobs and 155 scams, with 34 legitimate jobs misclassified as scams and 18 scams as legitimate jobs.

Random Forest Model Evaluation

	precision	recall	f1-score	support
0	0.89	0.80	0.84	174
1	0.82	0.90	0.86	173
accuracy			0.85	347
macro avg	0.85	0.85	0.85	347
weighted avg	0.85	0.85	0.85	347

Fig 18: Classification report of Random forest

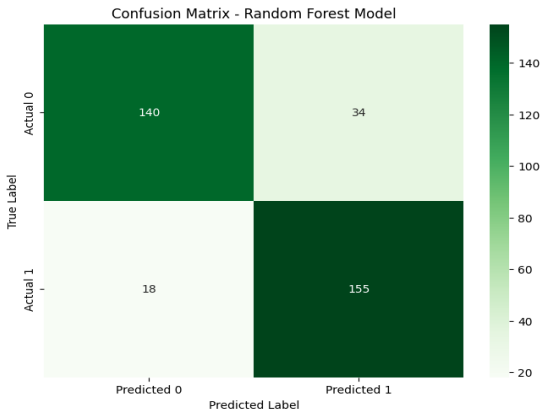

Fig 19: Confusion Matrix of Random forest

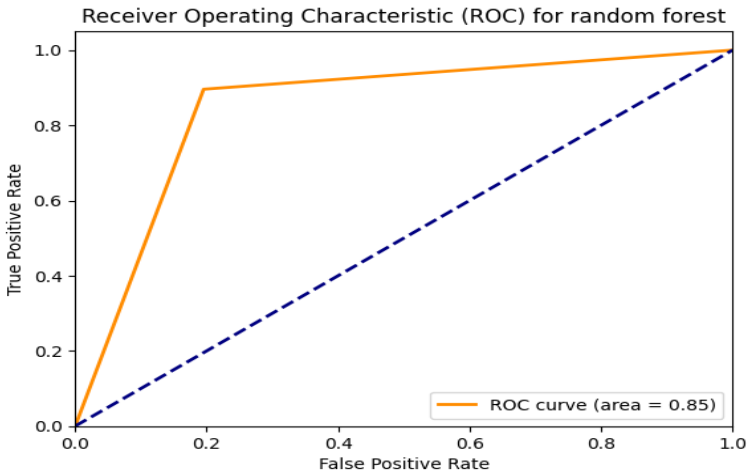

Fig 20: ROC for Random Forest

Comparison of Results for Different Models

Accuracy:

- Random Forest leads with 85% accuracy, followed by Logistic Regression (84%), KNN (78%), and Naive Bayes (66%).

Precision and Recall:

- Naive Bayes shows high recall (98% for scams) but lower precision, suggesting sensitivity towards detecting scams but with more false positives.
- KNN and Logistic Regression display more balanced precision and recall, indicating a better overall performance in correctly classifying both classes.
- Random Forest stands out with high precision and recall for both classes, demonstrating robustness.

F1-Score:

- Logistic Regression and Random Forest exhibit higher F1-scores, suggesting a more balanced detection capability.
- Naive Bayes and KNN have lower F1-scores, indicating some trade-offs in precision and recall.

Overall Analysis:

- Random Forest and Logistic Regression offer the best balance between detecting scams and reducing false positives.
- KNN performs well but lags slightly behind in precision and recall.
- Naive Bayes, while excellent in identifying scams, suffers from a high false positive rate, necessitating cautious use.

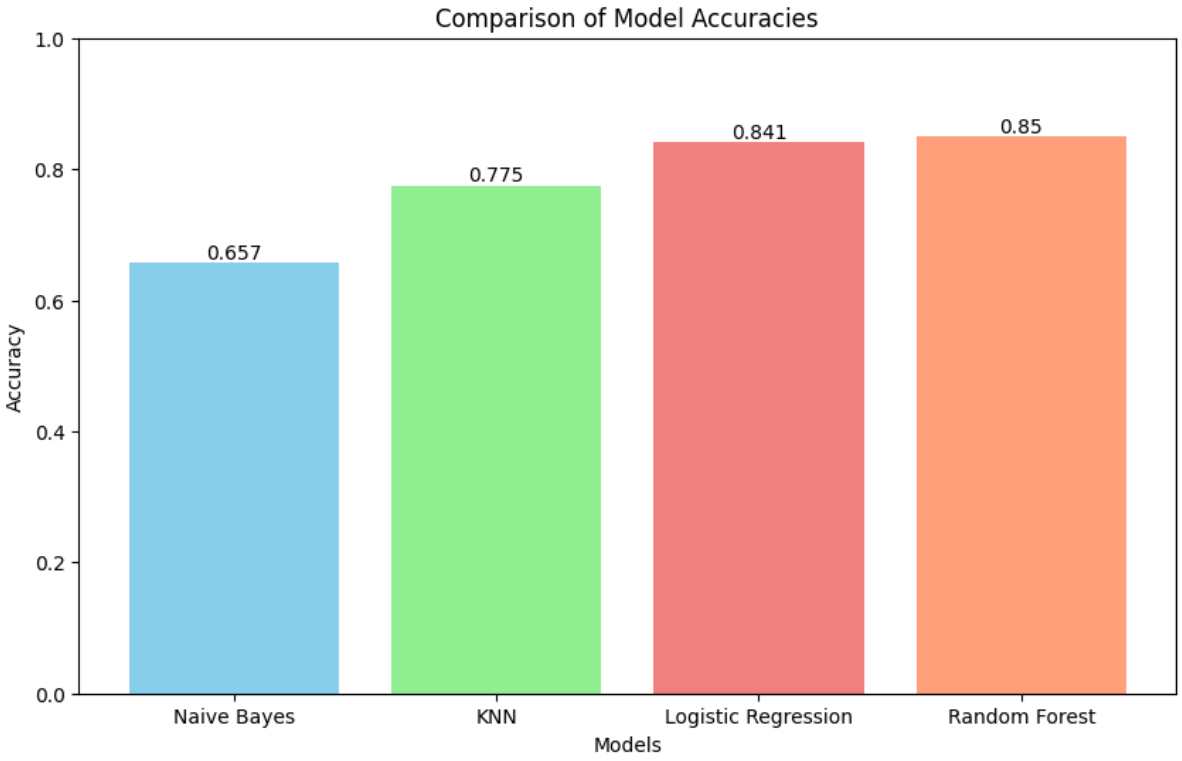

Fig 21: Comparison of Model Accuracies

Discussion and Future Work

Implement more sophisticated Natural Language Processing Techniques such as Named Entity Recognition (NER) and sentiment analysis to gain deeper insights into job descriptions. Also, Mobile applications can be developed so that when a person gives a job posting details in the application, it gives a result if it is a real or fake job posting.

Conclusion

In conclusion, this study presents a comprehensive analysis of job scam detection using machine learning techniques. The research encompassed data preparation, exploratory analysis, and the deployment of four distinct models: Naive Bayes, KNN, Logistic Regression, and Random Forest. Each model's effectiveness was rigorously evaluated using metrics like accuracy, precision, recall, and F1-score. The findings revealed that while each model has its strengths, Random Forest and Logistic Regression demonstrated the most balanced performance in detecting job scams. This study contributes to the growing field of data science and fraud detection, offering insights and methodologies that can be applied in similar contexts. Future work could explore the integration of these models or the use of more advanced machine learning techniques for enhanced accuracy and reliability.

References

[1] A. Amaar, W. Aljedaani, F. Rustam, S. Ullah, V. Rupapara, and S. Ludi, “Detection of fake job postings by utilizing machine learning and natural language processing approaches,” Neural Processing Letters, pp. 1–29, 2022.

[2] S. Dutta and S. K. Bandyopadhyay, “Fake job recruitment detection using machine learning approach,” International Journal of Engineering Trends and Technology, vol. 68, no. 4, pp. 48–53, 2020.

[3] S. Moon, M.-Y. Kim, and D. Iacobucci, “Content analysis of fake consumer reviews by survey-based text categorization,” International Journal of Research in Marketing, vol. 38, no. 2, pp. 343–364, 2021.

[4] M. Naud´e, K. J. Adebayo, and R. Nanda, “A machine learning approach to detecting fraudulent job types,” AI & SOCIETY, pp. 1–12, 2022.

[5] S. Bandyopadhyay and S. Dutta, “Fake job recruitment detection us- ing machine learning approach,” International Journal of Engineering Trends and Technology, vol. 68, 04 2020

Appendix - I

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.tokenize import sent_tokenize
from nltk.stem import WordNetLemmatizer
from collections import Counter
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import ConfusionMatrixDisplay

from google.colab import drive
drive.mount('/content/drive')

# loading the file containing the dataset and creating a dataframe of the dataset
df =pd.read_csv('/content/drive/My Drive/datasets/fake_job_postings.csv')

# display few records of the dataset
df.head(10)

# check the number of rows and columns of the dataset
df.dtypes

# check the number of rows and columns of the dataset
df.shape

# display the statistics of the numerical columns
df.describe()

# display the number of unique values each column has
unique_counts = df.nunique()
print(unique_counts)

# columns that have only 2 values - 0 and 1
```

```
binary_columns = ['telecommuting', 'has_company_logo', 'has_questions']

# display the unique values each column has
columns = ['title', 'location', 'department', 'salary_range', 'telecommuting', 'has_company_logo',
'has_questions', 'employment_type', 'required_experience',
        'required_education', 'industry', 'function']

for column in columns:
    print(f"{column}")
    print(df[column].value_counts())
    print()

# columns that have only 2 values - 0 and 1
binary_columns = ['telecommuting', 'has_company_logo', 'has_questions']

# plotting the countplots for the binary columns
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(12, 5))
sns.countplot(x='telecommuting', data=df, ax=axes[0])
axes[0].set_title('Telecommuting')

sns.countplot(x='has_company_logo', data=df, ax=axes[1])
axes[1].set_title('Company logo')

sns.countplot(x='has_questions', data=df, ax=axes[2])
axes[2].set_title('Questions')

plt.tight_layout()
plt.show()

# plotting the countplots for the columns- employment_type and required_experience
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(16, 5))
sns.countplot(x='employment_type', data=df, ax=axes[0])
axes[0].set_title('Employment type')

sns.countplot(x='required_experience', data=df, ax=axes[1])
axes[1].set_title('Required experience')

plt.tight_layout()
plt.show()
```

```
# plotting the countplots for the column- function
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(15, 5))
sns.countplot(x = 'function', data = df, order = df['function'].value_counts()[:10].index, ax=axes)
axes.set_title('Function')

plt.tight_layout()
plt.show()
print(df['fraudulent'].value_counts())

# visualizing the class imbalance using piechart
labels = ['Not fraudulent', 'Fraudulent']
sizes = [df['fraudulent'].value_counts()[0], df['fraudulent'].value_counts()[1]]
plt.pie(sizes, labels=labels, autopct='%1.1f%%', colors = ['lightgreen', 'yellow'])
plt.axis('equal')
plt.show()

# checking for NA values in the dataset
df.isna().sum()

# checking for duplicate records in the dataset
df.duplicated().sum()

# dropping the 'job_id' column as it is irrelevant to the task at hand
df.drop(columns=['job_id'],inplace=True)

# displaying the correlation heatmap to check for multicollinear features
corr = df.corr()
sns.heatmap(corr, annot=True)
plt.show()
df['location']

# creating a function to retrieve only the country from the 'location' column. This column
originally contains the country and cities as well. The retrieved countries are then inserted into a
ne3w column 'country'.
def split(location):
    if pd.notna(location):
        address = location.split(',')
        return address[0]
    else:
        return None
```

```
df['country'] = df['location'].apply(split)

# displaying unique values of the retrieved countries and their counts
df['country'].value_counts()

# plotting the countplot for the new column- country
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(7, 5))
sns.countplot(x = 'country', data = df, order = df['country'].value_counts()[:5].index, ax=axes)
axes.set_title('Country')
plt.tight_layout()
plt.show()

# check number of unique countries
country_counts = df['country'].nunique()
print(country_counts)

# these columns contain NA values.
cols_tofill=['title','location', 'department', 'salary_range', 'company_profile','description',
'requirements', 'benefits',
        'employment_type','required_experience','required_education','industry','function']

# Replacing the NA values with None
df[cols_tofill] = df[cols_tofill].fillna(value='None')

# After replacement, there are no NA values
df.isna().sum()
df

# Creating two new dataframes where the fakejobs dataframe contains only fradulent job posting
records whereas the realjobs dataframe contains only real job posting records.
fakejobs = df[df['fraudulent']==1]
realjobs = df[df['fraudulent']==0]

# plotting the countplots for columns- country, employment_type, required_experience, industry,
and function for both the dataframes separately
fig,axes=plt.subplots(5,2,figsize=(30,50))
sns.countplot(x = 'country', data = fakejobs, palette='Set2',order =
fakejobs['country'].value_counts()[:5].index, ax=axes[0,0])
```

```
sns.countplot(x = 'country', data = realjobs, palette='Set2',order =
realjobs['country'].value_counts()[:5].index,ax=axes[0,1])
axes[0,0].set_title('Highest 5 Countries of Fake Jobs',fontsize=15)
axes[0,1].set_title('Highest 5 Countries of Real Jobs',fontsize=15)
axes[0,0].set_xlabel('Country')
axes[0,1].set_xlabel('Country')

sns.countplot(x = 'employment_type', data = fakejobs, palette='Set2',order =
fakejobs['employment_type'].value_counts()[:5].index,ax=axes[1,0])
sns.countplot(x = 'employment_type', data = realjobs, palette='Set2',order =
realjobs['employment_type'].value_counts()[:5].index,ax=axes[1,1])
axes[1,0].set_title('Fake Jobs Most Employment Types',fontsize=15)
axes[1,1].set_title('Real Jobs Most Employment Types',fontsize=15)
axes[1,0].set_xlabel('Employment Type')
axes[1,1].set_xlabel('Employment Type')

sns.countplot(x = 'required_experience', data = fakejobs, palette='Set2',order =
fakejobs['required_experience'].value_counts()[1:5].index,ax=axes[2,0])
sns.countplot(x = 'required_experience', data = realjobs, palette='Set2',order =
realjobs['required_experience'].value_counts()[1:5].index,ax=axes[2,1])
axes[2,0].set_title('Fake Jobs Most Required Experience',fontsize=15)
axes[2,1].set_title('Real Jobs Most Required Experience',fontsize=15)
axes[2,0].set_xlabel('Required Experience')
axes[2,1].set_xlabel('Required Experience')

sns.countplot(x = 'industry', data = fakejobs, palette='Set2',order =
fakejobs['industry'].value_counts()[1:5].index,ax=axes[3,0])
sns.countplot(x = 'industry', data = realjobs, palette='Set2',order =
realjobs['industry'].value_counts()[1:5].index,ax=axes[3,1])
axes[3,0].set_title('Fake Jobs Most Industries',fontsize=15)
axes[3,1].set_title('Real Jobs Most Industries',fontsize=15)
axes[3,0].set_xlabel('Industryy')
axes[3,1].set_xlabel('Industry')

sns.countplot(x = 'function', data = fakejobs, palette='Set2',order =
fakejobs['function'].value_counts()[1:5].index,ax=axes[4,0])
sns.countplot(x = 'function', data = realjobs, palette='Set2',order =
realjobs['function'].value_counts()[1:5].index,ax=axes[4,1])
axes[4,0].set_title('Fake Jobs Most Functions',fontsize=15)
axes[4,1].set_title('Real Jobs Most Functions',fontsize=15)
```

```
axes[4,0].set_xlabel('Function')
axes[4,1].set_xlabel('Function')

# plotting histograms for the character count of 'description' for both dataframes
fig,axes=plt.subplots(1,2,figsize=(10,5))
df[df['fraudulent']==0]['description'].str.len().plot(bins=15, kind='hist', color='blue', label='Real',
alpha=0.8, ax=axes[0])
df[df['fraudulent']==1]['description'].str.len().plot(bins=10, kind='hist', color='red', label='Fake',
alpha=0.8, ax=axes[1])
fig.suptitle('Character count in Description of Job posting')
axes[0].set_xlabel("Character Count")
axes[0].set_ylabel("Frequency")
axes[0].legend()

axes[1].set_xlabel("Character Count")
axes[1].set_ylabel("Frequency")
axes[1].legend()

plt.tight_layout()
plt.show()

# plotting histograms for the character count of 'requirements' for both dataframes
fig,axes=plt.subplots(1,2,figsize=(10,5))
df[df['fraudulent']==0]['requirements'].str.len().plot(bins=15, kind='hist', color='blue',
label='Real', alpha=0.8, ax=axes[0])
df[df['fraudulent']==1]['requirements'].str.len().plot(bins=10, kind='hist', color='red',
label='Fake', alpha=0.8, ax=axes[1])
fig.suptitle('Character count in Requirements of Job posting')
axes[0].set_xlabel("Character Count")
axes[0].set_ylabel("Frequency")
axes[0].legend()

axes[1].set_xlabel("Character Count")
axes[1].set_ylabel("Frequency")
axes[1].legend()

plt.tight_layout()
plt.show()

# plotting histograms for the character count of 'benefits' for both dataframes
```

```
fig,axes=plt.subplots(1,2,figsize=(10,5))

df[df['fraudulent']==0]['benefits'].str.len().plot(bins=15, kind='hist', color='blue', label='Real',
alpha=0.8, ax=axes[0])
df[df['fraudulent']==1]['benefits'].str.len().plot(bins=10, kind='hist', color='red', label='Fake',
alpha=0.8, ax=axes[1])
fig.suptitle('Character count in Benefits of Job posting')
axes[0].set_xlabel("Character Count")
axes[0].set_ylabel("Frequency")
axes[0].legend()

axes[1].set_xlabel("Character Count")
axes[1].set_ylabel("Frequency")
axes[1].legend()

plt.tight_layout()
plt.show()

# plotting histograms for the character count of 'company_profile' for both dataframes
fig,axes=plt.subplots(1,2,figsize=(10,5))

df[df['fraudulent']==0]['company_profile'].str.len().plot(bins=15, kind='hist', color='blue',
label='Real', alpha=0.8, ax=axes[0])
df[df['fraudulent']==1]['company_profile'].str.len().plot(bins=10, kind='hist', color='red',
label='Fake', alpha=0.8, ax=axes[1])
fig.suptitle('Character count in Company profile of Job posting')
axes[0].set_xlabel("Character Count")
axes[0].set_ylabel("Frequency")
axes[0].legend()

axes[1].set_xlabel("Character Count")
axes[1].set_ylabel("Frequency")
axes[1].legend()

plt.tight_layout()
plt.show()

# plotting histograms for the word count of 'description' for both dataframes
fig,axes=plt.subplots(1,2,figsize=(10,5))
```

```
df[df['fraudulent']==0]['description'].apply(lambda x: len(str(x).split())).plot(bins=20, kind='hist',
color='blue', label='Real', alpha=0.8, ax=axes[0])
df[df['fraudulent']==1]['description'].apply(lambda x: len(str(x).split())).plot(bins=10, kind='hist',
color='red', label='Fake', alpha=0.8, ax=axes[1])
fig.suptitle('Word count in Description of Job posting')
axes[0].set_xlabel("Word Count")
axes[0].set_ylabel("Frequency")
axes[0].legend()

axes[1].set_xlabel("Word Count")
axes[1].set_ylabel("Frequency")
axes[1].legend()

plt.tight_layout()
plt.show()

# plotting histograms for the word count of 'requirements' for both dataframes
fig,axes=plt.subplots(1,2,figsize=(10,5))

df[df['fraudulent']==0]['requirements'].apply(lambda x: len(str(x).split())).plot(bins=20,
kind='hist', color='blue', label='Real', alpha=0.8, ax=axes[0])
df[df['fraudulent']==1]['requirements'].apply(lambda x: len(str(x).split())).plot(bins=10,
kind='hist', color='red', label='Fake', alpha=0.8, ax=axes[1])
fig.suptitle('Word count in Requirements of Job posting')
axes[0].set_xlabel("Word Count")
axes[0].set_ylabel("Frequency")
axes[0].legend()

axes[1].set_xlabel("Word Count")
axes[1].set_ylabel("Frequency")
axes[1].legend()

plt.tight_layout()
plt.show()

# plotting histograms for the word count of 'benefits' for both dataframes
fig,axes=plt.subplots(1,2,figsize=(10,5))

df[df['fraudulent']==0]['benefits'].apply(lambda x: len(str(x).split())).plot(bins=20, kind='hist',
color='blue', label='Real', alpha=0.8, ax=axes[0])
```

```
df[df['fraudulent']==1]['benefits'].apply(lambda x: len(str(x).split())).plot(bins=10, kind='hist', color='red', label='Fake', alpha=0.8, ax=axes[1])
fig.suptitle('Word count in Benefits of Job posting')
axes[0].set_xlabel("Word Count")
axes[0].set_ylabel("Frequency")
axes[0].legend()

axes[1].set_xlabel("Word Count")
axes[1].set_ylabel("Frequency")
axes[1].legend()

plt.tight_layout()
plt.show()

# plotting histograms for the word count of 'company_profile' for both dataframes
fig,axes=plt.subplots(1,2,figsize=(10,5))

df[df['fraudulent']==0]['company_profile'].apply(lambda x: len(str(x).split())).plot(bins=15, kind='hist', color='blue', label='Real', alpha=0.8, ax=axes[0])
df[df['fraudulent']==1]['company_profile'].apply(lambda x: len(str(x).split())).plot(bins=10, kind='hist', color='red', label='Fake', alpha=0.8, ax=axes[1])
fig.suptitle('Word count in Company profile of Job posting')
axes[0].set_xlabel("Word Count")
axes[0].set_ylabel("Frequency")
axes[0].legend()

axes[1].set_xlabel("Word Count")
axes[1].set_ylabel("Frequency")
axes[1].legend()

plt.tight_layout()
plt.show()

#Combining all the text columns
df['text'] =  df['title'] + ' ' + df['country'] + ' ' + df['company_profile'] + ' ' +  df['description'] + ' ' + df['requirements'] + ' ' + \
            df['benefits'] + ' ' + df['required_experience'] + ' ' + df['required_education'] + ' ' + df['industry'] + ' ' + df['function']
#Display first element of text column
df['text'][0]
```

```
#Change text column to list
text = df.text.to_list()

#display first element of text
text[0]

#length of text
len(text)

#download required libraries from nltk
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('stopwords')
lemmatizer = WordNetLemmatizer()
from wordcloud import WordCloud

# Join the text elements into a single string
cloud_text = ' '.join(map(str,text))

# Tokenize the text
token_text_cloud = word_tokenize(cloud_text)

# Convert tokens to lowercase
lower_tokens_cloud = [t.lower() for t in token_text_cloud]

# Count the frequency of each word
word_freq_cloud = Counter(lower_tokens_cloud)
print(Counter.most_common(word_freq_cloud, 10))
text_only_alphabets_cloud = [t for t in lower_tokens_cloud if t.isalpha()]
english_stopped_cloud = stopwords.words('english')
no_stops_cloud = [t for t in text_only_alphabets_cloud if t not in english_stopped_cloud]

# Lemmatize the words
lemmatized_words_cloud = [lemmatizer.lemmatize(t) for t in no_stops_cloud]
bow = Counter(lemmatized_words_cloud)
print(Counter.most_common(bow, 10))

# Filter out words that are 'None' or empty strings after lemmatization
```

```
lemmatized_words_without_none = [word for word in lemmatized_words_cloud if word and
word.lower() != 'none']
# Combine the filtered words into a single string
all_words = ' '.join(lemmatized_words_without_none)
# Create a WordCloud object with specified parameters
wordcloud = WordCloud(width=800, height=500, random_state=21,
max_font_size=120).generate(all_words)
plt.figure(figsize=(10, 8))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
# Show the plot
plt.show()
#change text to string of sentance
text = [str(sentence) for sentence in text]
# Initialize an empty list to store tokenized sentences
token_text = []
# Iterate over each sentence in the 'text' list
for sentence in text:
    tokens = word_tokenize(sentence)
    token_text.append(tokens)
# Create a list comprehension to convert each word to lowercase in each sentence
lower_tokens = [[t.lower() for t in tokens] for tokens in token_text]
#create a list of only alphabets
text_only_alphabets = [[t for t in tokens if t.isalpha()] for tokens in lower_tokens]
nltk.download('stopwords')
# Fetch the list of English stopwords using NLTK
stopwords.words('english')
# Fetch the list of English stopwords using NLTK
english_stopped = stopwords.words('english')
# Remove stopwords from each list of tokens in 'text_only_alphabets'
no_stops = [[t for t in tokens if t not in english_stopped] for tokens in text_only_alphabets]
# Create a WordNetLemmatizer object for lemmatizing words
lemmatizer = WordNetLemmatizer()
# Lemmatize each word in each list of tokens in 'no_stops'
lemmatized_words = [[lemmatizer.lemmatize(t) for t in tokens] for tokens in no_stops]
lemmatized_words[0]
# Filter out words that are 'None' or empty strings after lemmatization
final_text = [
    [word for word in tokens if word and word.lower() != 'none']
    for tokens in lemmatized_words
```

```
]
#drop all the columns that were combied in text column
drop_columns = ['title','country','company_profile', 'description', 'requirements',
'benefits','required_experience', 'required_education', 'industry', 'function' ]
df = df.drop(columns =drop_columns)
df['text']=final_text
df
df['text'] = df['text'].apply(lambda tokens: ' '.join(tokens))

from imblearn.under_sampling import RandomUnderSampler
import pandas as pd

# Instantiate RandomUnderSampler
rus = RandomUnderSampler(random_state=42)


X = df.drop('fraudulent', axis=1)
y = df['fraudulent']

#resample the data
X_resampled, y_resampled = rus.fit_resample(X, y)


resampled_data = pd.concat([pd.Series(y_resampled), pd.DataFrame(X_resampled)], axis=1)
resampled_data.columns = ['fraudulent'] + list(X.columns)

# Define a list of categorical columns
categorical_columns = [ 'location', 'department', 'salary_range',  'employment_type', 'text']
#categorical_columns = [ 'department', 'text']
resampled_data = pd.get_dummies(resampled_data, columns=categorical_columns)

#plot the pie chart for resampled data
labels = ['Not fraudulent', 'Fraudulent']
sizes = [resampled_data['fraudulent'].value_counts()[0],
resampled_data['fraudulent'].value_counts()[1]]
plt.pie(sizes, labels=labels, autopct='%1.1f%%', colors = ['lightgreen', 'yellow'])
plt.axis('equal')
plt.show()
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score,confusion_matrix,f1_score,classification_report
from sklearn.metrics import precision_score,recall_score
from sklearn.metrics import roc_curve, auc

#assign training values
y = resampled_data['fraudulent']
X = resampled_data.loc[:,resampled_data.columns!='fraudulent']
print(X.shape)
X.head()

#train the data for modelling
x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42,stratify=y)

Navie bayes
#modelling using Gussian Naive Bayes
nb = GaussianNB()
nb.fit(x_train, y_train)
#predict the values for Naive Bayes
y_pred_nb = nb.predict(x_test)
print("Navie Bayes Model Evaluation")
print(classification_report(y_test, y_pred_nb))
# Assuming y_test and y_pred are already defined
Cfm_nb = confusion_matrix(y_test, y_pred_nb)

# Display the confusion matrix
print("Confusion Matrix for Naive Bayes Model:")
print(Cfm_nb)

# If you want to visualize the confusion matrix using a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(Cfm_nb, annot=True, fmt="d", cmap="Oranges", xticklabels=['Predicted 0',
'Predicted 1'], yticklabels=['Actual 0', 'Actual 1'])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix - Naive Bayes Model")
plt.show()
#calculate roc
fpr, tpr, thresholds = roc_curve(y_test, y_pred_nb)
```

```
roc_auc = auc(fpr, tpr)
print(roc_auc)
#plot the graph for ROC
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
#display the graph
plt.show()
```

KNN

```
# Fit a KNN model on the training data
k = 2 # Set the number of neighbors to consider
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(x_train, y_train)
# Predict the target variable on the testing data
y_pred_knn = knn.predict(x_test)
print("KNN Model Evaluation")
print(classification_report(y_test, y_pred_knn))

# Assuming y_test and y_pred are already defined
Cfm_knn = confusion_matrix(y_test, y_pred_knn)

# Display the confusion matrix
print("Confusion Matrix for KNN Model:")
print(Cfm_knn)

# If you want to visualize the confusion matrix using a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(Cfm_knn, annot=True, fmt="d", cmap="Purples", xticklabels=['Predicted 0',
'Predicted 1'], yticklabels=['Actual 0', 'Actual 1'])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix - KNN Model")
plt.show()
```

```
# Calculate the ROC curve and area under the curve (AUC)
fpr, tpr, thresholds = roc_curve(y_test, y_pred_knn)
roc_auc = auc(fpr, tpr)
print("ROC : ", roc_auc)
# Plot the ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
#Display ROC graph
plt.show()
```

LOGISTIC REGRESSION

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report,accuracy_score

log_reg = LogisticRegression()
log_reg.fit(x_train, y_train)
y_pred_log = log_reg.predict(x_test)
print("Logistic Regression Model Evaluation")
print(classification_report(y_test, y_pred_log))
# Assuming y_test and y_pred_log are already defined
cm_log = confusion_matrix(y_test, y_pred_log)

# Display the confusion matrix
print("Confusion Matrix for Logistic Regression Model:")
print(cm_log)

# If you want to visualize the confusion matrix using a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm_log, annot=True, fmt="d", cmap="Blues", xticklabels=['Predicted 0', 'Predicted
1'], yticklabels=['Actual 0', 'Actual 1'])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
```

```
plt.title("Confusion Matrix - Logistic Regression Model")
plt.show()

#calculate ROC
fpr, tpr, thresholds = roc_curve(y_test, y_pred_log)
roc_auc = auc(fpr, tpr)
print(roc_auc)
#plot ROC graph
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) for logistic regression')
plt.legend(loc="lower right")
#display ROC graph
plt.show()
```

RANDOM FOREST

```
#random forest classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
#train random forest
rf_classifier.fit(x_train, y_train)
y_pred_rf = rf_classifier.predict(x_test)
print("Random Forest Model Evaluation")
#display values
print(classification_report(y_test, y_pred_rf))
# Assuming y_test and y_pred_rf are already defined
cm_rf = confusion_matrix(y_test, y_pred_rf)

# Display the confusion matrix
print("Confusion Matrix for Random Forest Model:")
print(cm_rf)

# If you want to visualize the confusion matrix using a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm_rf, annot=True, fmt="d", cmap="Greens", xticklabels=['Predicted 0', 'Predicted
1'], yticklabels=['Actual 0', 'Actual 1'])
```

```
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix - Random Forest Model")
plt.show()
#ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_rf)
roc_auc = auc(fpr, tpr)
print(roc_auc)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) for random forest')
plt.legend(loc="lower right")
#show ROC
plt.show()

**COMPARISION OF MODELS**
# Assuming y_test, y_pred_nb, y_pred_knn, y_pred_log, and y_pred_rf are already defined

# Calculate accuracies for each model
accuracy_nb = accuracy_score(y_test, y_pred_nb)
accuracy_knn = accuracy_score(y_test, y_pred_knn)
accuracy_log = accuracy_score(y_test, y_pred_log)
accuracy_rf = accuracy_score(y_test, y_pred_rf)

# Model names
models = ['Naive Bayes', 'KNN', 'Logistic Regression', 'Random Forest']

# Corresponding accuracies
accuracies = [accuracy_nb, accuracy_knn, accuracy_log, accuracy_rf]

# Create a bar plot
plt.figure(figsize=(10, 6))
bars = plt.bar(models, accuracies, color=['skyblue', 'lightgreen', 'lightcoral', 'lightsalmon'])
plt.ylim(0, 1)  # Set y-axis range to represent accuracy percentage (0-1)
plt.title('Comparison of Model Accuracies')
```

```
plt.xlabel('Models')
plt.ylabel('Accuracy')

# Add annotations to display accuracies on the bars
for bar, accuracy in zip(bars, accuracies):
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval, round(accuracy, 3), ha='center', va='bottom',
color='black', fontsize=10)

plt.show()
```