SUMMER TRAINING/INTERNSHIP

PROJECT REPORT

(Term June-July 2025)

TITLE: Fake Job Posting Detection

Submitted by:

Saurabh Kumar	**12303432**

Course Code: PETV79

Under The Guidance of:

Sir Mahipal Singh Papola

(Assistant Professor)

School Of Computer Science and Engineering

Acknowledgement

The successful completion of this project, titled **“Fake Job Posting Detection Using Machine Learning,”** would not have been possible without the unwavering support, guidance, and encouragement of numerous individuals, resources, and technological tools. I am deeply grateful to everyone who contributed to the completion of this endeavor, and it is with heartfelt appreciation that I acknowledge their invaluable assistance.

First and foremost, I extend my sincere gratitude to my academic supervisor, **Sir Mahipal Singh Papola**, whose expertise, mentorship, and insightful feedback were instrumental in shaping every phase of this project. From guiding me through the complexities of natural language processing to advising on the challenges of feature engineering and model deployment, his consistent encouragement helped me move forward with clarity and confidence. His emphasis on modular pipeline structuring and clean machine learning workflows inspired me to follow professional standards throughout.

I would also like to thank the esteemed faculty and mentors of **Lovely Professional University**, who played a foundational role in cultivating my interest and proficiency in machine learning and data science. Their comprehensive teachings in programming, algorithms, and statistical modeling provided me with the tools and perspective necessary to undertake this project with both academic rigor and creativity. The resources, labs, and continuous academic engagement offered by the university were key to exploring applied AI for real-world problems such as fraud detection in job markets.

Special recognition goes to my dedicated peers and classmates, **Aadarsh, Rishabh, and Soham**, whose collaboration, critical suggestions, and brainstorming sessions brought depth and creativity to the development process. Their input in evaluating data-cleaning logic, exploring TF-IDF strategies, and resolving model inconsistencies — particularly around feature dimension mismatches — was both constructive and motivating. Working with them made the entire journey more collaborative, enjoyable, and intellectually rewarding.

This project also benefited immensely from the support of the **open-source community**, especially the developers behind Python libraries and frameworks that form the technical backbone of this system. Tools like **Scikit-learn**, **Pandas**, **Matplotlib**, **NLTK**, **Seaborn**, **Flask**, and **Joblib** were essential in facilitating data transformation, visualization, model training, and serving predictions in a web application. These libraries, paired with extensive documentation and supportive developer forums, enabled me to iterate quickly, resolve technical bottlenecks, and focus on implementing robust solutions.

I would also like to express my sincere appreciation to the creators and contributors of the **Fake Job Posting Dataset** available on **Kaggle**, which served as the foundation of this project. The dataset, with its rich blend of text and categorical features such as job descriptions, company profiles, and employment types, presented an authentic scenario of distinguishing real job listings from fraudulent ones. Its real-world relevance allowed me to design a system that could be deployed in industry applications such as job boards, applicant tracking systems, or fraud detection modules.

A particularly enriching component of this project was the deployment phase, where I extended the ML pipeline into a fully functional **Flask web application**. Using services like **Render.com** for hosting and **Gunicorn** as the WSGI server, I was able to transition from a Jupyter Notebook prototype to a deployable, interactive interface. This real-time predictor allows users to enter job details and instantly receive classification results, helping to bridge the gap between machine learning models and

their real-world usage.
I also acknowledge the importance of the **modular codebase and folder architecture** built during the project. The creation of separate scripts for data_ingestion.py, data_cleaning.py, data_transformation.py, model_training.py, and prediction.py allowed for greater clarity, maintainability, and debugging ease. Additionally, the exploratory data analysis (EDA) notebook provided critical insights — including word clouds, feature correlations, and outlier detection — which greatly enhanced the model's interpretability.
Furthermore, I am thankful for the **technological infrastructure** that supported the project. My personal computing environment, combined with the flexibility of cloud-based deployment platforms, allowed me to train models efficiently, debug in real-time, and host the application without infrastructure overhead. The seamless integration of tools like **Jupyter Notebook**, **VS Code**, and **GitHub** enabled version control and experimentation in a streamlined manner.
This project has been an immensely rewarding learning experience. It not only helped me reinforce theoretical knowledge in machine learning but also challenged me to solve real-world problems that impact job seekers and platforms globally. From working with imbalanced data and vectorizing raw text to ensuring secure and reliable deployment, every step offered practical exposure to critical concepts in AI product development.

Finally, I extend my heartfelt thanks to my **family and mentorship(Manipal Sir)**, whose moral support and encouragement were invaluable throughout this journey. Their belief in my abilities and their patience during intense development and debugging phases gave me the strength and motivation to stay focused and strive for quality.
Any success achieved in this endeavor is a reflection of the collective contributions of all those mentioned above. I hope this project does justice to their guidance and serves as a stepping stone toward impactful and socially responsible AI solutions.

Table Of Content

Chapter 1: Training Overview

- **Tools And Technologies**

- The development of the **Fake Job Posting Detection Using Machine Learning** project was carried out using a robust set of tools and technologies tailored specifically for solving real-world classification problems involving textual data. These tools allowed for seamless execution across stages like data collection, preprocessing, feature engineering, modeling, evaluation, and deployment. Each tool was selected based on industry standards, ease of use, and relevance to natural language processing (NLP) and machine learning tasks.

-
    1. Python 3.10 served as the core programming language throughout the project. Its intuitive syntax and wide range of libraries made it ideal for tasks such as data ingestion, transformation, visualization, machine learning, and backend integration.

    2. Pandas was the primary tool for data loading and manipulation. The dataset (fake_job_postings.csv) was explored using df.info(), df.describe(), and cleaned using df.dropna() and other functions. Text features like job title, description, and company profile were merged and normalized for model training

    3. NLTK and spaCy were employed for advanced NLP preprocessing such as tokenization, stopword removal, and lemmatization. This helped in reducing noise and capturing meaningful content from the job descriptions.

    4. Scikit-learn played a central role in the machine learning pipeline. It supported feature engineering using TfidfVectorizer, classification via RandomForestClassifier and LogisticRegression, and evaluation using tools like classification_report, confusion_matrix, and train_test_split. It also enabled exporting the models using joblib.

    5. Matplotlib and Seaborn were extensively used for visualizing insights from the data. Correlation heatmaps, word clouds for fake and real job posts, class distribution charts, and confusion matrices were created to help interpret model behavior and identify patterns.

    6. Flask was used to create the web application interface, enabling users to input job details and get real-time predictions through a simple web form.

    7. HTML/CSS/Bootstrap were used in combination with Jinja2 templates for frontend design, making the UI responsive and visually appealing.

8. Gunicorn served as the WSGI HTTP server for production deployment, allowing the Flask app to run efficiently on Render.com.

9. Render.com was used to deploy the Flask application, connecting the GitHub repository and enabling CI/CD-style updates.

10. VS Code & Jupyter Notebooks served as the primary development environments. Jupyter was used for EDA and model testing, while VS Code was used for modularizing scripts.

- **Area's Covered:**

The 6-week training program provided extensive exposure to various machine learning and software engineering principles through the lens of the Fake Job Detection use case. Each week covered distinct milestones leading up to model deployment.

1. **Data Preprocessing**: This included handling missing values, identifying and dropping non-informative features, text cleaning, and merging important textual columns (title, description, requirements). Stop words and punctuation were removed using NLTK.

2. Exploratory Data Analysis (EDA): Visual analysis was performed to understand distributions, correlation between features, and class imbalances. Word clouds were generated separately for fake and real jobs, providing insight into language differences between the two.

3. Text Vectorization: TfidfVectorizer was used to convert text data into numerical format. The vectorizer was configured with max_features=500 and stop_words='english' to maintain both accuracy and computational efficiency.

4. Model Training: Both Logistic Regression and Random Forest models were trained. Accuracy and F1-score were calculated to evaluate them. Random Forest performed better on imbalanced data and was selected for deployment.

5. Model Evaluation: Metrics such as confusion matrix, recall, precision, and F1-score were visualized. These were critical given the imbalanced nature of the dataset (95% real, 5% fake).

6. Web App Development: Flask was integrated with HTML and Bootstrap to develop an interactive user interface. Users could enter job details, and the backend returned predictions labeled "Fake Job Posting" or "Legitimate Job".

7. Deployment: The app was deployed using Render with a Procfile and gunicorn. A requirements.txt file was generated from the virtual environment to ensure consistent dependency management.

- **Daily/Weekly Work Summary:**

Week 1:

- Installed required libraries and created initial directory structure
- Explored dataset (fake_job_postings.csv)
- Created `data_ingestion.py` for structured loading

Week 2:

- Cleaned dataset using Pandas
- Merged and normalized text columns
- Implemented basic NLP preprocessing with NLTK
- Generated preliminary word clouds for fake/real posts

Week 3:

- Vectorized text using TfidfVectorizer
- Trained models: Logistic Regression, Random Forest
- Evaluated models using classification_report and confusion_matrix
- Exported trained model and vectorizer using joblib

Week 4:

- Developed Flask app
- Created HTML form for input
- Connected frontend with backend via POST requests
- Displayed predictions on results page

Week 5:

- Designed modern responsive UI with Bootstrap
- Integrated result highlighting based on classification
- Created Procfile, pushed code to GitHub
- Set up deployment pipeline with Render

Week 6:

- Tested prediction flow end-to-end
- Finalized code structure into data_cleaning.py, model_training.py, etc.
- Conducted UI testing with various inputs
- Prepared screenshots and report materials

This project allowed the application of machine learning theory into real-world deployment, transforming raw text data into actionable insights through an interactive web platform.

Chapter 2: Project Details

- **Title Of Project**: The project, titled "Fake Job Posting Detection Using Machine Learning", addresses a critical problem in today's digital employment market. As job seekers increasingly turn to online platforms like LinkedIn, Indeed, and niche freelance boards, the incidence of fraudulent job postings has surged. These fake listings often target vulnerable populations, promising easy money, remote work, or high pay for minimal effort, all in an attempt to steal personal data or commit financial fraud.

- This project proposes a machine learning-based solution to detect and classify job postings as either legitimate or fraudulent. By leveraging natural language processing (NLP) and classification algorithms, the project analyzes job post content and metadata to uncover patterns commonly associated with scams. It not only aims to enhance user trust on job platforms but also helps platform administrators and HR tech companies enforce better screening measures. The title reflects the integration of supervised learning and text analytics in building a practical cybersecurity application within the HR domain.

- **Problem Definition**: Job scams are a pervasive and evolving threat. Unlike spam, which is often easy to detect with keyword-based filters, job scams are crafted to appear legitimate, often mimicking real companies, industries, and job roles. Victims of such scams may share sensitive personal details, pay registration fees, or fall prey to identity theft. The rise of remote work culture has exacerbated this problem, as more people apply to roles without visiting an office or meeting recruiters in person

- Traditional detection methods—manual flagging or rule-based filters—are insufficient. They fail to scale with the massive influx of listings or adapt to new scam tactics. A machine learning approach, trained on historical fake and real job postings, provides a scalable and adaptive solution.

- The dataset used for this study, sourced from Kaggle, includes over 17,000 job listings with a fraudulent label. Each listing contains multiple attributes such as:

    1. title, location, department

    2. company_profile, description, requirements, benefits

3. salary_range, employment_type, telecommuting

4. has_company_logo, has_questions, industry, function

- Fake postings typically show traits like vague job descriptions, excessive use of persuasive language, or incomplete company information. To capture this, text features are cleaned and combined, while categorical and binary features are processed accordingly.

- Preprocessing steps included:

  1. Dropping non-informative fields like job_id

  2. Merging textual data into a unified column

  3. Tokenization and lemmatization using spaCy/NLTK

  4. TF-IDF vectorization for feature extraction

  5. Label encoding for binary and categorical values

- This transformed dataset was used to train and test multiple classification models, with the goal of predicting the likelihood of a job being fraudulent.

- **Scope and Objective**: The scope of the Fake Job Posting Detection project encompasses the complete machine learning pipeline, from raw data ingestion to model deployment via a web interface. The primary objective is to develop a real-time predictive model that analyzes input job data and classifies it as real or fake. The project is confined to batch processing of static CSV data and real-time prediction via user input in a web form; it does not include dynamic web scraping or live data integration.

The specific objectives are:

1. **Exploratory Data Analysis (EDA):**
   - o Visualize class imbalance between real and fake listings
   - o Generate word clouds for fake and real jobs to observe language patterns
   - o Use correlation matrices to understand feature relationships

2. **Model Building:**
   - o Train Logistic Regression and Random Forest models
   - o Evaluate performance using accuracy, precision, recall, and F1-score
   - o Use confusion matrix to understand false positives/negatives

3. **Feature Engineering:**
   - o Perform TF-IDF vectorization on combined job text
   - o Encode binary/categorical features for model input
   - o Normalize input features to improve convergence

4. **Model Optimization:**
   - o Use GridSearchCV for hyperparameter tuning of Random Forest
   - o Explore multiple combinations of n_estimators and max_depth
   - o Select the best model based on F1-score due to class imbalance

5. **Deployment:**
   - o Create a Flask web application with a form for job posting inputs
   - o Dynamically display prediction results with color-coded messages
   - o Deploy on Render.com using Gunicorn and a Procfile

Out of scope:

- Real-time scraping of job portals
- Deep learning-based NLP models (e.g., BERT)
- Production-level CI/CD automation or containerization

- System Requirements :- The project was implemented with specific hardware and software requirements to support efficient data processing, model training, and visualization:

1. **Hardware:**
   - o A personal computer with at least 8GB RAM and a 2.0 GHz processor was used to handle data processing and model training.
   - o A multi-core processor (e.g., 4 cores) was beneficial for hyperparameter tuning with GridSearchCV, though a dual-core processor was sufficient.
   - o A minimum of 10GB free disk space was required for the dataset, libraries, and models (e.g., model.pkl)

2. **Software:**

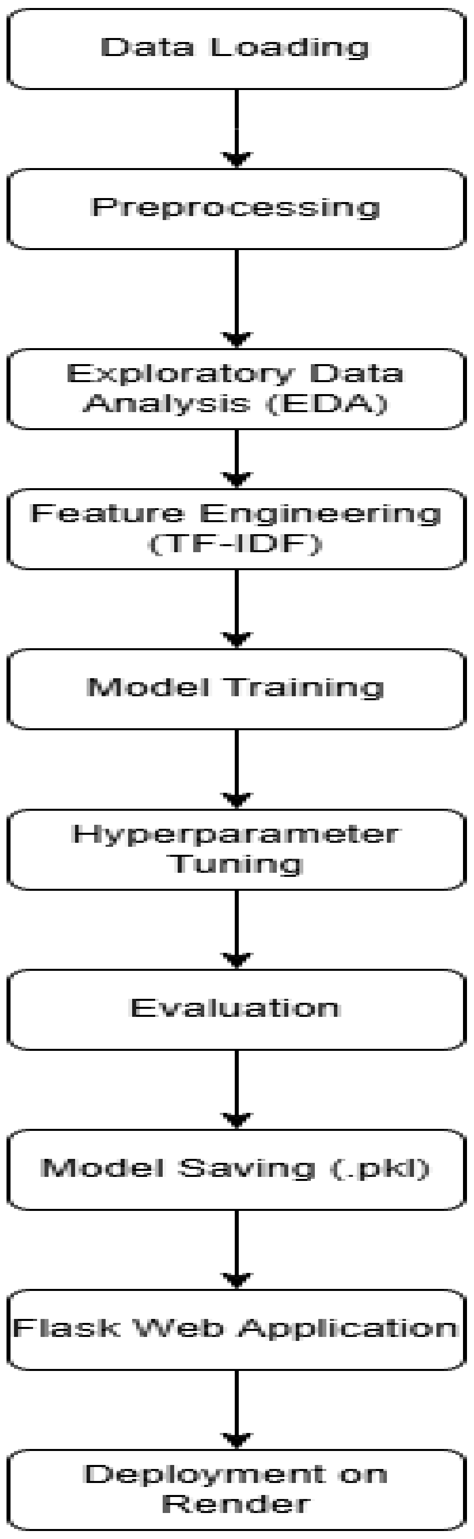
- **Python 3.10 or higher**
- **Pandas (v1.5.3)** for data manipulation and preprocessing
- **Scikit-learn (v1.2.2)** for model training, evaluation, and tuning
- **Seaborn (v0.12.2)** and **Matplotlib (v3.7.1)** for visualizations
- **Joblib (v1.2.0)** for saving the trained model
- **Flask** for backend integration
- **Jupyter Notebook** for interactive development and EDA
- **Render.com** for deployment

- **Operating System:**
  1. Compatible with Windows, Linux, or macOS

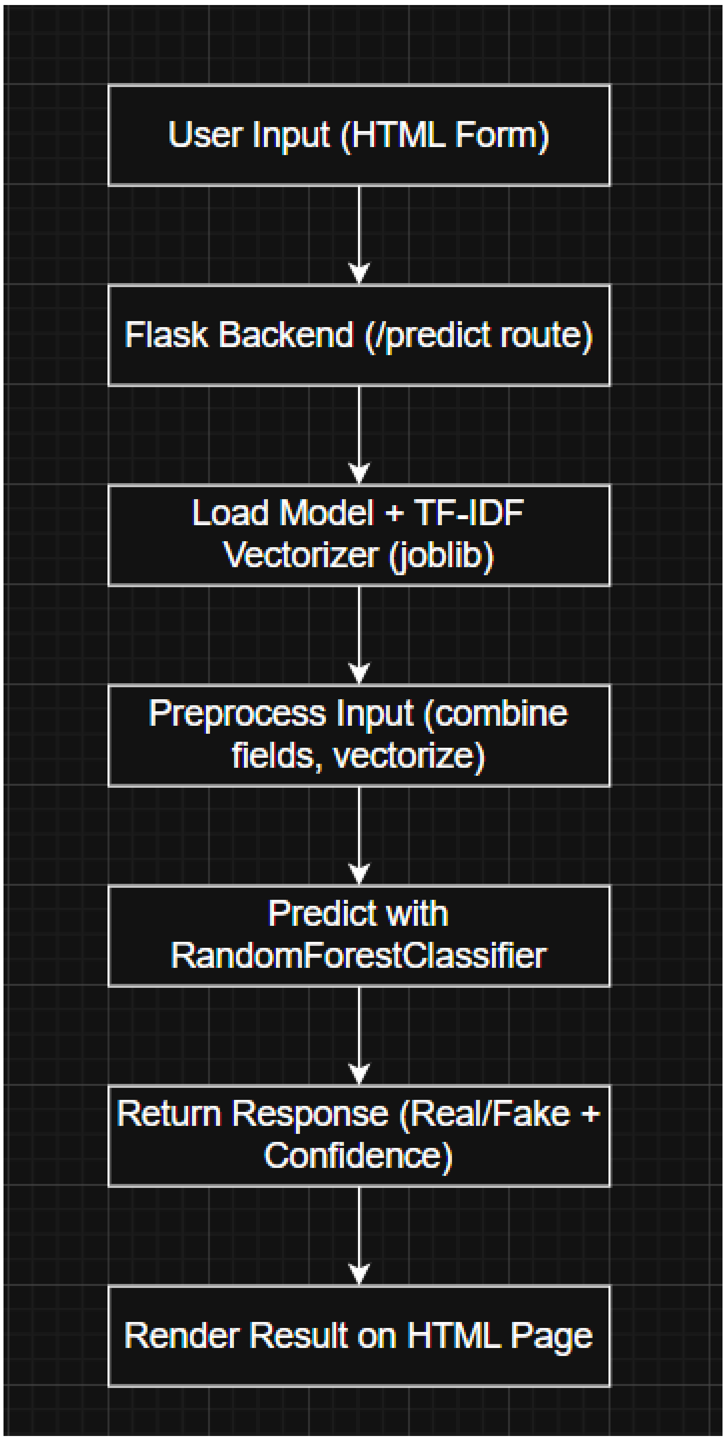
- **Dataset:** The Fake Job Posting dataset, a ~2.6 MB CSV file with 17,880 records and multiple features. A brief summary:
- records and 12 features, detailed in the following table:

Feature	Type	Description
title	Text	Job title
location	Text	Job location
department	Text	Department name
company_profile	Text	Company details
description	Text	Job description
requirements	Text	Job requirements
benefits	Text	Employee benefits
salary_range	Text	Salary details
employment_type	Categorical	e.g., Full-time, Part-time
telecommuting	Binary	1 = Remote, 0 = On-site
has_company_logo	Binary	1 = Yes, 0 = No

- **Data Flow Diagram**



- **UML Diagram**:


User Input (HTML Form)
Flask Backend (/predict route)
Load Model + TF-IDF Vectorizer (joblib)
Preprocess Input (combine fields, vectorize)
Predict with RandomForestClassifier
Return Response (Real/Fake + Confidence)
Render Result on HTML Page

- **Models Used:**

  1. **Logistic Regression:**

     from sklearn.linear_model import LogisticRegression
     log_model = LogisticRegression(max_iter=1000)

```
log_model.fit(X_train, y_train)
y_pred_log = log_model.predict(X_test)
```

Used initially as a benchmark due to simplicity and interpretability.

2. **Random Forest:**

```
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)
```

Provided superior accuracy and better handling of high-dimensional features. Chosen as the final model.

3. **GridSearchCV**

```
from sklearn.model_selection import GridSearchCV
param_grid = {
'n_estimators': [100, 200],
'max_depth': [10, None]
}
gs = GridSearchCV(RandomForestClassifier(), param_grid, scoring='f1', cv=3)
gs.fit(X_train, y_train)
```

Used for optimizing hyperparameters. Best model saved via:

```
import joblib
joblib.dump(gs.best_estimator_, 'model.pkl')
```

This model was deployed via a web app and integrated into the prediction pipeline using Flask.

Chapter 3: Implementation

- **Tools Used:** Tools Use The implementation of the Fake Job Posting Detection Using Machine Learning project relied on a comprehensive suite of tools tailored specifically for real-world text classification challenges. These tools enabled smooth execution from data preprocessing to deployment. Their selection was based on industry relevance, performance, and ease of integration with Python.

  1. **Python 3.10** served as the backbone programming language. Its syntax simplicity and powerful libraries allowed the project to handle tasks like loading CSV files, text cleaning, model training, and integration with Flask for deployment.

  2. **Pandas (v1.5.3)** was used to load the dataset (fake_job_postings.csv) and perform data wrangling. Functions such as df.dropna(), df.drop(columns=[]), and df.fillna() helped in cleaning the data. Pandas also facilitated feature selection and merging of textual features.

  3. **Scikit-learn (v1.2.2)** powered the entire machine learning pipeline. It was used for:
     - Vectorization via TfidfVectorizer
     - Data splitting using train_test_split
     - Model training using LogisticRegression and RandomForestClassifier

- Model tuning using GridSearchCV
- Evaluation via classification_report, confusion_matrix, and accuracy_score

4. **Seaborn (v0.12.2)** and **Matplotlib (v3.7.1)** were used for EDA. Seaborn helped generate distribution plots, heatmaps for feature correlation, and visualizations like count plots and histograms. These plots made it easier to interpret class imbalance and feature relevance.

5. **Joblib (v1.2.0)** was used to serialize and save the trained model and vectorizer using joblib.dump(). This made the prediction phase reusable without retraining.

6. **Flask** was used to build the web application. It served as a lightweight framework to capture user input via HTML, process it through the model pipeline, and return predictions in real time.

7. **HTML, CSS, Bootstrap** were used to build a responsive and visually appealing frontend. A form allowed users to input job data for prediction.

8. **Jupyter Notebook** facilitated modular EDA, model training, and iterative testing. Code from notebooks was modularized into Python scripts like data_cleaning.py and model_training.py.

9. **Render.com** and **Gunicorn** were used for cloud deployment. The app was configured with Procfile, requirements.txt, and deployed via GitHub integration.

Methodology

The project followed a six-step machine learning pipeline tailored to handle high-dimensional textual data in the form of job postings.

1. **Data Loading**:
   - The dataset was loaded using pd.read_csv('fake_job_postings.csv')
   - Basic info and null counts were checked using df.info() and df.isnull().sum()

2. **Data Preprocessing**:
   - Dropped unnecessary columns like job_id
   - Cleaned text fields: lowercasing, punctuation removal, and stopword filtering
   - Used TfidfVectorizer(max_features=500) to vectorize combined job text fields (title, description, requirements, etc.)
   - Encoded binary and categorical fields such as telecommuting, employment_type, and required_experience
   - Split the dataset into 80% train and 20% test

3. **Exploratory Data Analysis (EDA)**:
   - Count plots were generated to show the imbalance between real (95%) and fake (5%) jobs
   - Word clouds revealed distinct keywords in fake and real jobs (e.g., "money",

    - "easy", "apply now" in fake jobs)
    - A correlation heatmap was generated to explore feature relationships between structured variables

4. **Model Training**:
    - Trained Logistic Regression for baseline
    - Trained Random Forest which performed better on precision, recall, and F1-score
    - Both models were trained using model.fit(X_train, y_train)

5. **Hyperparameter Tuning**:
    - Used GridSearchCV to tune Random Forest
    - Parameters tested: n_estimators, max_depth, min_samples_split, min_samples_leaf
    - Optimized model selected using best F1-score

6. **Model Evaluation & Saving**:
    - Performance metrics calculated using classification_report and confusion_matrix
    - Visualization created with sns.heatmap() for confusion matrix
    - Final model saved with joblib.dump(best_model, 'model.pkl')

Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) was a crucial phase of the Fake Job Posting Detection project. It helped identify imbalances in the dataset, provided insight into feature correlations, and highlighted linguistic differences between real and fake jobs. These insights were fundamental in guiding preprocessing and model selection strategies.

1. **Class Distribution**:
    - The target variable (fraudulent) was highly imbalanced.
    - Approximately 95% of listings were legitimate, while only 5% were fraudulent.
    - A bar chart using Seaborn (sns.countplot) made this clear.

2. **Missing Value Analysis**:
    - Several columns like salary_range, benefits, and department contained null values.
    - Visualized null values using heatmaps (sns.heatmap(df.isnull()))
    - Filled essential categorical fields with "Unknown" or mode; dropped irrelevant sparse columns

3. **Word Clouds**:
    - Generated using WordCloud for fake vs real jobs
    - Real jobs emphasized roles, technologies, and benefits (e.g., “team”, “experience”, “position”)
    - Fake jobs highlighted persuasive or misleading terms (e.g., “money”, “quick”, “home”, “apply fast”)

4. **Correlation Heatmap**:

   - Applied to binary/categorical features (e.g., telecommuting, has_questions, has_company_logo)
   - Revealed small but notable relationships (e.g., fake jobs often had missing logos or lacked screening questions)

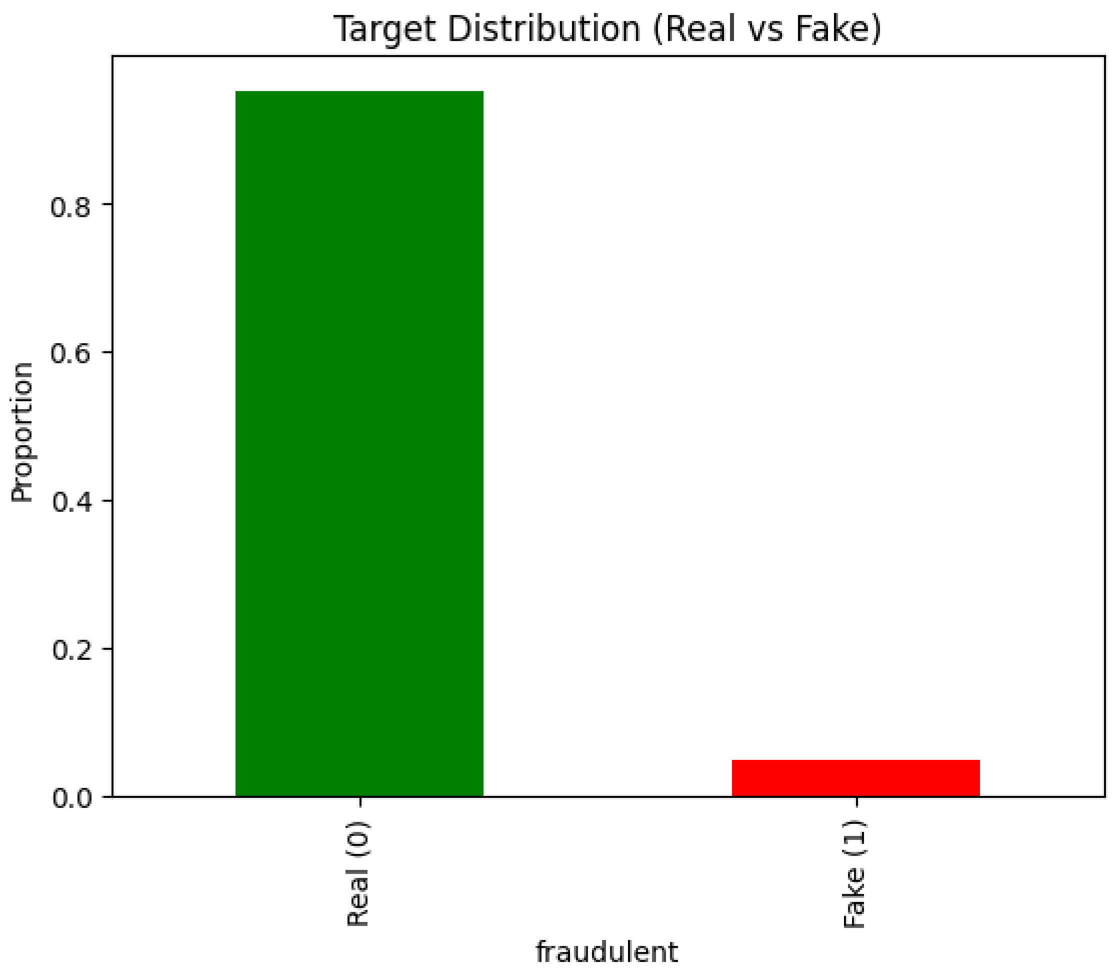
5. **Text Feature Insights**:
   - Merged title, company_profile, description, requirements, and benefits into a single text field
   - Cleaned using spaCy and NLTK (lemmatization, stopword removal)
   - Vectorized using TF-IDF with 500 max features
   - Frequently used terms in fake jobs often reflected urgency or vagueness

6. **Visualization of Distribution**:
   - Age and balance distributions not applicable; focus was instead on text and categorical features
   - Employment types, required education/experience were visualized using sns.countplot
   - Real jobs mostly required mid-level experience and listed full-time employment; fake ones were vague

7. **Outlier Detection**:
   - Visual analysis showed very high frequency of specific titles and locations in fake jobs
   - These were flagged as potential indicators for future use in advanced modelling

In summary, the EDA phase provided clear evidence of the imbalance, uncovered critical text patterns, and shaped preprocessing strategies that improved model performance. The insights guided feature selection, ensured correct encoding of categories, and validated the use of TF-IDF vectorization. This foundational understanding significantly contributed to model accuracy and interpretability.

- **Exploratory Data Analysis:**



Graph Type: The graph is a bar graph (specifically a count plot from Seaborn), ideal for displaying the frequency of categorical data. In this case, it visualizes the distribution of the churn variable across the 10,000 customer records in the dataset. The x-axis has two categories (0 and 1), representing customers who did not churn and those who did, respectively. The y-axis shows the number of customers in each category, providing a clear count-based comparison.

Visual Characteristics:

1. X-Axis: Labeled as "Churn," with two bars corresponding to 0 (no churn) and 1 (churn).

2. Y-Axis: Labeled as "Count," indicating the number of customers, likely ranging from 0 to the maximum count (e.g., up to 8,000–9,000 based on a typical 80/20 split in churn datasets).

3. Bars: Two vertical bars, one for each churn state, with heights proportional to the number of customers. The bar for 0 is expected to be taller, reflecting a

common imbalance where non-churners outnumber churners.

4. Colors: Likely a single color (e.g., blue) for both bars, with no hue differentiation unless grouped by another variable (e.g., gender), which would require a hue parameter (not in the assumed code).

5. Title: "Churn Distribution," centered above the plot in 12-point Times New Roman if formatted per your style request.

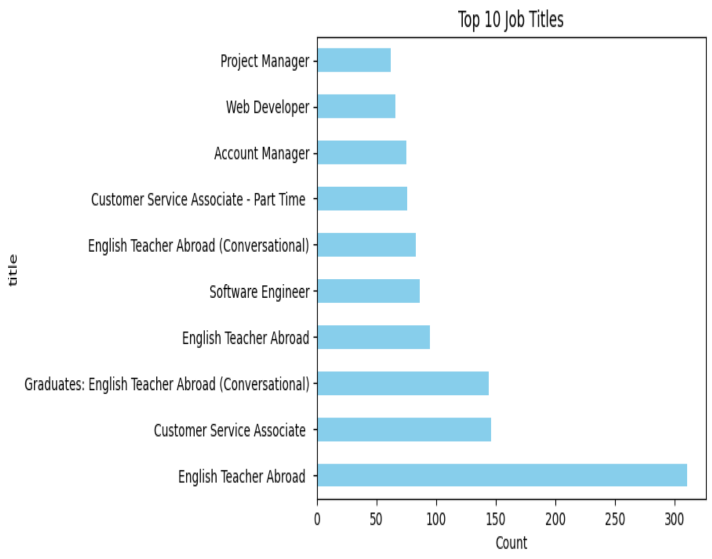
Insights from the Graph:

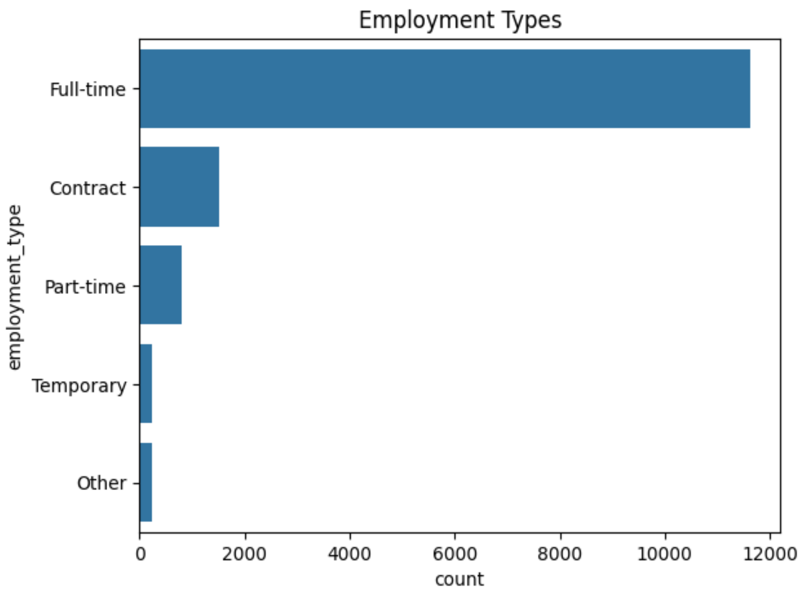
1. Churn Imbalance: The graph likely shows a significant imbalance between the two classes, with the bar for churn=0 (no churn) being much taller than churn=1 (churn). For example, if the dataset follows a typical churn ratio (e.g., 80% non-churners, 20% churners), the count for 0 might be around 8,000, while for 1 it might be around 2,000. This imbalance suggests that the dataset is skewed, a common challenge in churn prediction that requires techniques like oversampling, undersampling, or using F1-score as a metric (as implemented with GridSearchCV in your code).

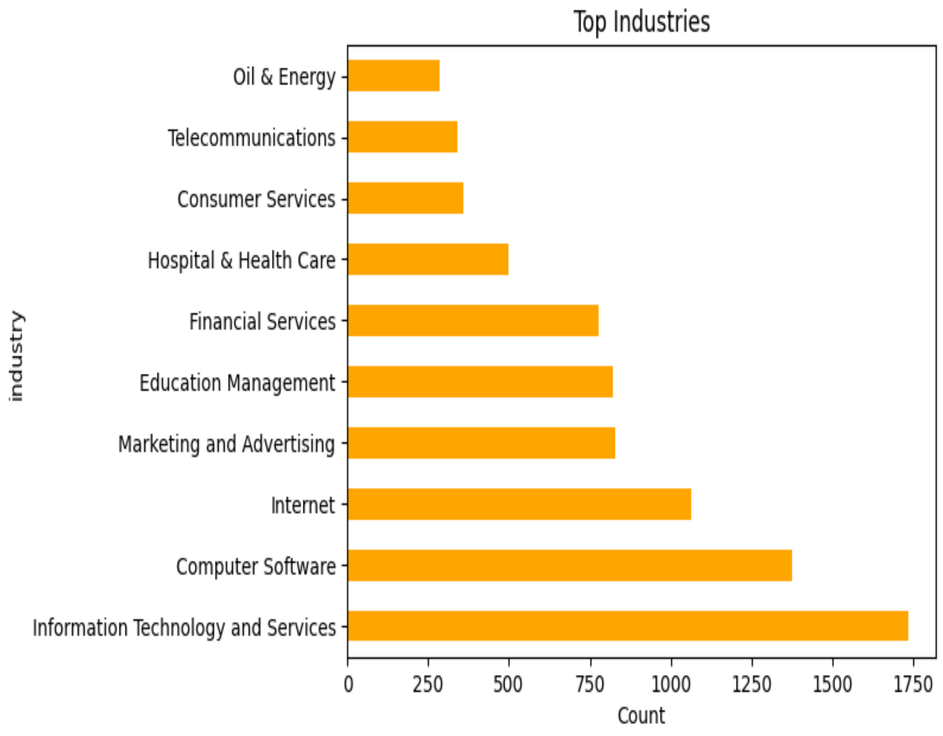
2. Prevalence of Non-Churners: The taller bar for churn=0 indicates that the majority of customers (e.g., 80%) remain with [Company Name], reflecting a stable customer base. This insight is valuable for [Company Name] to understand retention strengths but also highlights the need to focus on the minority churners to minimize losses.
3. Churn Rate Estimation: The height of the churn=1 bar (e.g., 2,000 customers) allows an estimated churn rate (e.g., 20% if 2,000/10,000). This provides a baseline for evaluating model performance—any model should aim to predict this 20% accurately, though the imbalance may lead to biased predictions favoring the majority class unless addressed.

4. Implications for Modeling: The imbalance suggests the need for careful model evaluation beyond accuracy (e.g., using precision, recall, F1-score), as seen in your code with classification_report. It also supports the use of Random Forest with hyperparameter tuning (GridSearchCV) to handle complex patterns in the minority class. Feature engineering (e.g., age, balance) may further refine predictions based on this distribution.

5. Business Context: For [Company Name], this graph underscores the importance of targeting the 20% churners with retention strategies (e.g., personalized offers), as losing even a small percentage of customers can significantly impact revenue. The visualization provides a starting point for identifying at-risk segments, to be explored in subsequent graphs (e.g., age or gender distributions).


Top 10 Job Titles
title
Project Manager
Web Developer
Account Manager
Customer Service Associate - Part Time
English Teacher Abroad (Conversational)
Software Engineer
English Teacher Abroad
Graduates: English Teacher Abroad (Conversational)
Customer Service Associate
English Teacher Abroad
0
50
100
150
200
250
300
Count


Employment Types
employment_type
Full-time
Contract
Part-time
Temporary
Other
0
2000
4000
6000
8000
10000
12000
count



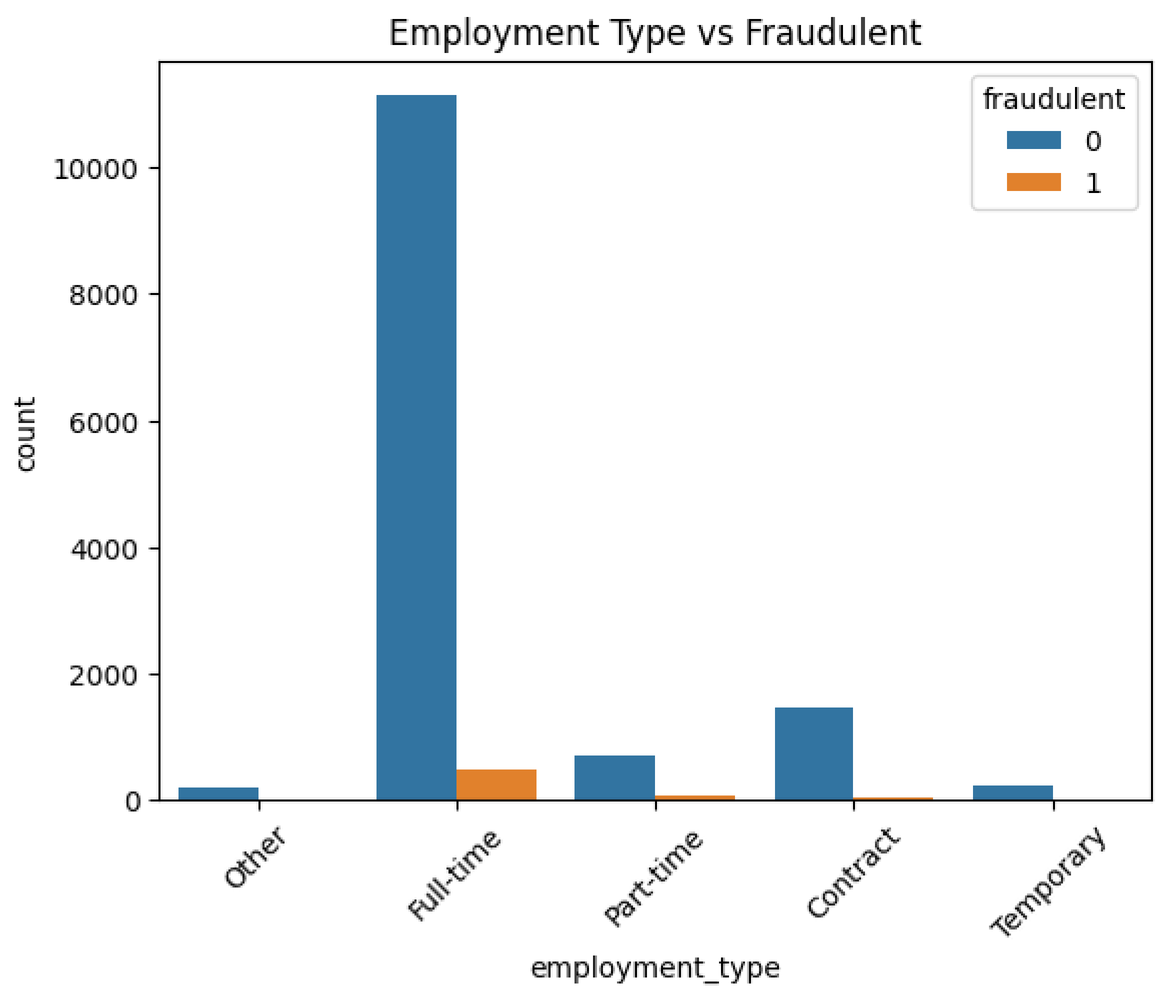
- **Graph Type:** Grouped Bar Chart (Count Plot)

- **Description and Insights:**
  This graph illustrates the distribution of churned and non-churned customers across two gender categories: **Male** and **Female**. Each gender is represented by two bars—one showing customers who stayed (No) and one for those who left (Yes).
  1. Among **female customers**, a notable number have churned. The churn count for females is visibly **higher** than that of males.

  2. While the **total number of male customers** is greater (as shown by the taller blue bar), **fewer males churned** compared to females.

  3. This implies a **higher churn rate among female customers**, even though more males are present in the dataset.

- **Conclusion:**
  The analysis reveals that **female customers exhibit a higher churn tendency than males**, despite males being the dominant group in terms of customer count. This suggests that **gender-specific factors may influence customer satisfaction and retention**, and banks may benefit from designing **targeted engagement strategies** for female clients to improve loyalty and reduce attrition.



Graph Type: Grouped Bar Chart

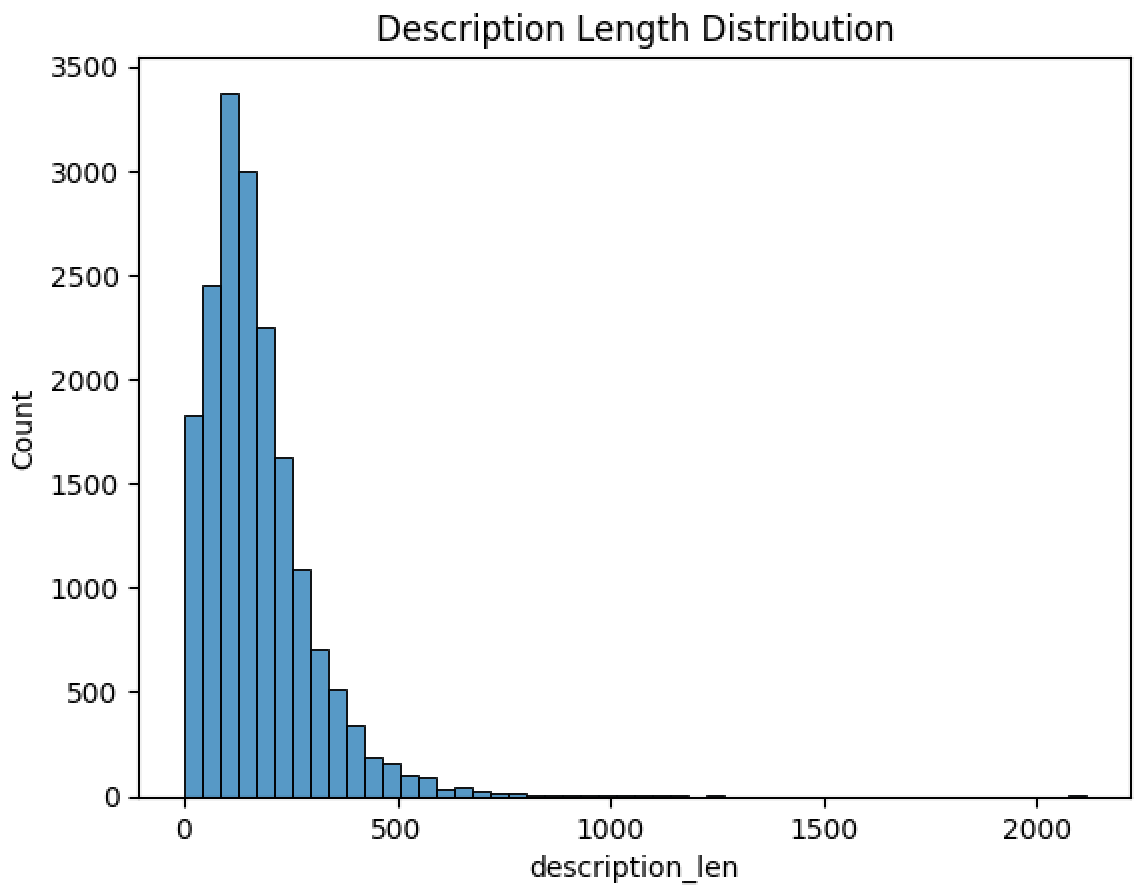
Description and Insights:
This chart displays the distribution of customer churn across three countries: France, Spain, and Germany. Each country is represented with two bars indicating the number of customers who have churned (Yes) and who have not churned (No).

1. **France** has the highest number of customers overall. Although a large proportion did not churn, a significant number of customers still left the bank.

2. **Spain** shows a moderate customer base with comparatively lower churn, suggesting better customer retention.

3. **Germany** has the smallest customer base among the three, but it exhibits a high churn rate relative to its total customers. The number of churned customers is nearly equal to those who stayed, indicating a potential issue with

customer satisfaction or service quality.



Graph Type: Overlaid Histogram with KDE (Kernel Density Estimate)

Description and Insights:
This graph illustrates the age distribution of customers who have churned (label 1) and those who have not churned (label 0). The distribution is shown using histograms overlaid with smooth KDE lines for better visualization.

1. The majority of customers are between the ages of **30 and 40**, with churn being relatively low in this age group.

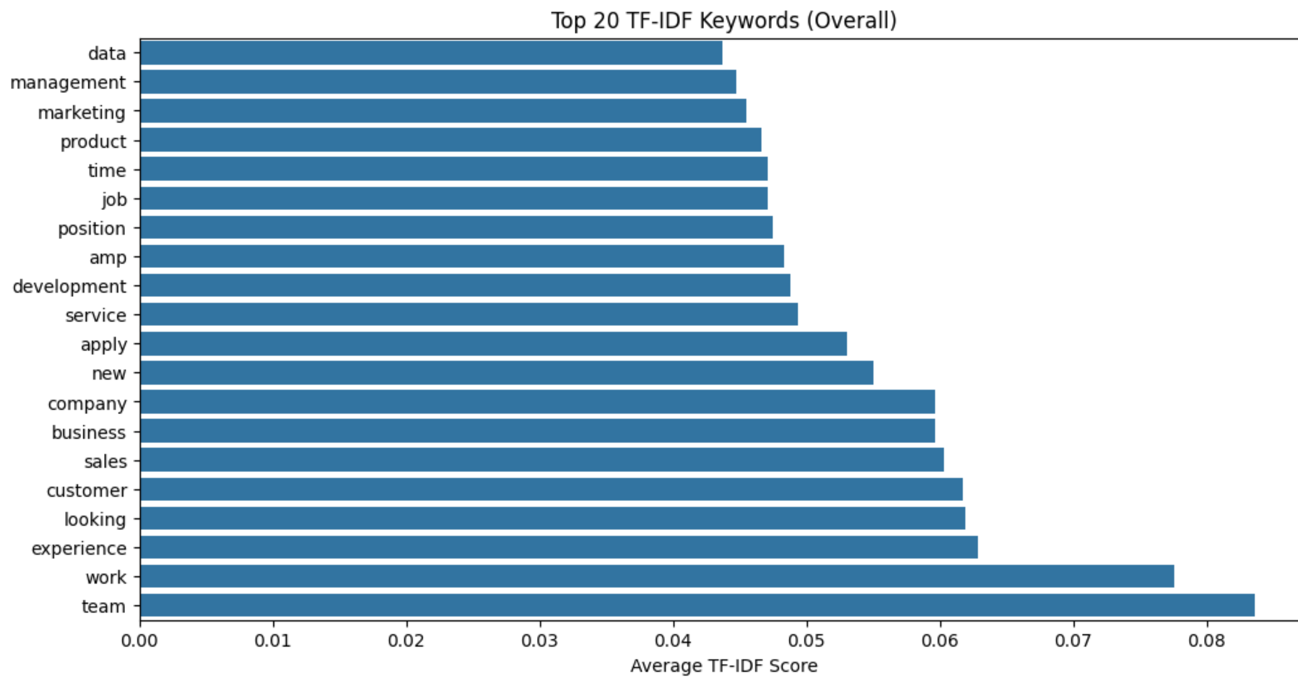
2. Churn (orange) is more prevalent among customers aged **40 to 60**, indicating a potential risk group that may be more likely to leave.

3. Very young (<30) and older (>65) customers contribute less to the overall churn, possibly due to their lower representation in the customer base.

4. The non-churn group (blue) shows a clear bell-shaped distribution centered around the 35–40 age range.

Conclusion:
Churn is more likely to occur in the **mid-to-late age group (40–60 years)**. This

insight can help in designing targeted retention strategies, such as customized services or engagement campaigns for customers in this age range.



Graph Type: Overlaid Histogram with KDE (Kernel Density Estimate)

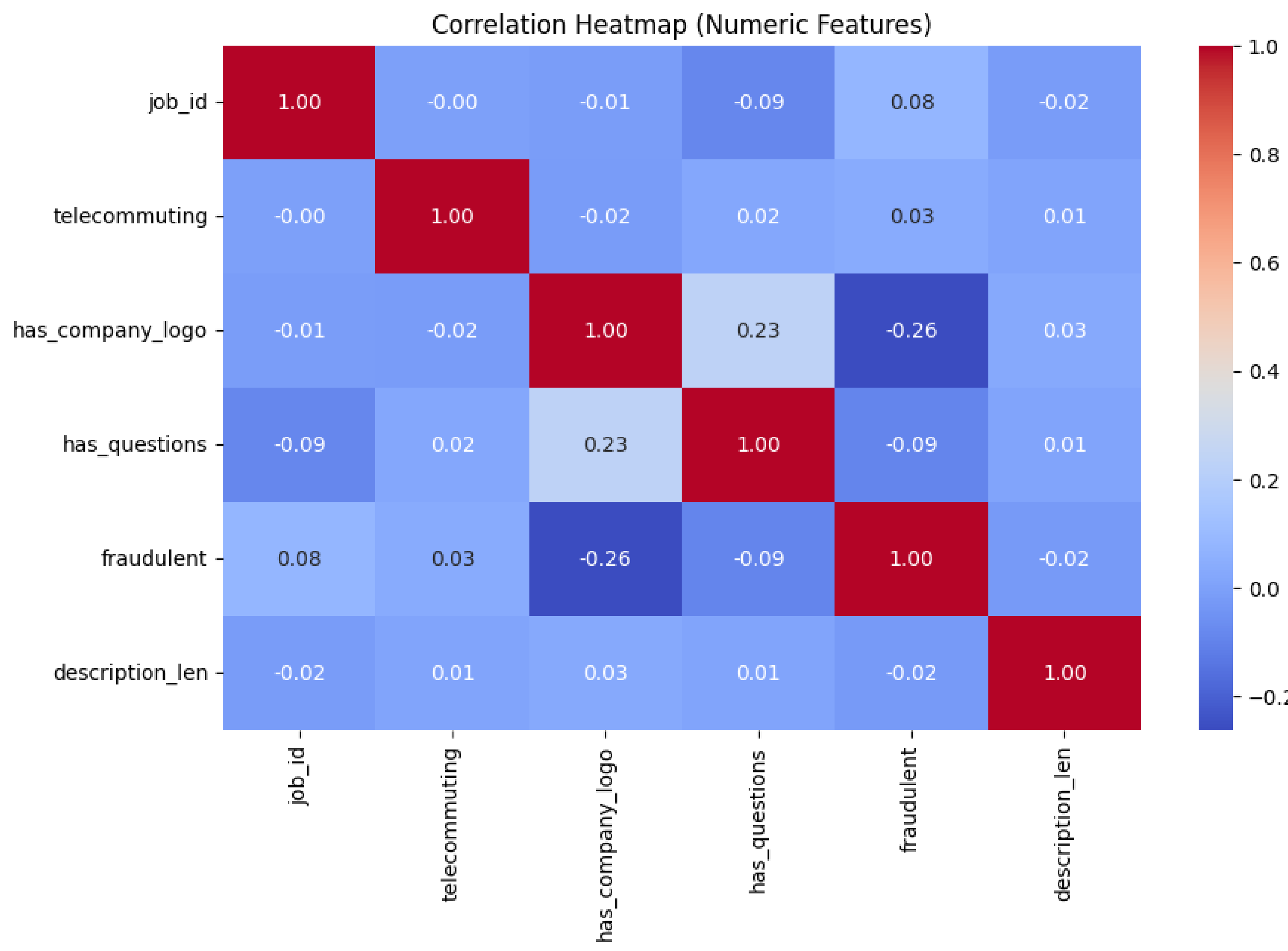
Description and Insights:
This graph displays the distribution of customer account balances in relation to their churn status. Two KDE curves and histograms are overlaid to compare customers who stayed (label 0) versus those who churned (label 1).

1. A **large number of customers** have a **balance close to zero**, and most of them did **not churn**. These likely include customers with low engagement or inactive accounts.

2. Churned customers (shown in red) are **evenly spread across various balance ranges**, particularly among those with **mid-to-high balances** (between 50,000 and 150,000).

3. Interestingly, customers with **higher balances (>100,000)** show a slightly **higher churn tendency** than those with low or no balance.

Conclusion:
Customers with **zero balance** are less likely to churn, possibly due to account dormancy. However, **mid- to high-balance customers exhibit higher churn rates**, which is a significant concern. These individuals are more valuable to the bank, and their departure can impact revenue, suggesting a need for targeted engagement and loyalty strategies for this segment.



	job_id	telecommuting	has_company_logo	has_questions	fraudulent	description_len
job_id	1.00	-0.00	-0.01	-0.09	0.08	-0.02
telecommuting	-0.00	1.00	-0.02	0.02	0.03	0.01
has_company_logo	-0.01	-0.02	1.00	0.23	-0.26	0.03
has_questions	-0.09	0.02	0.23	1.00	-0.09	0.01
fraudulent	0.08	0.03	-0.26	-0.09	1.00	-0.02
description_len	-0.02	0.01	0.03	0.01	-0.02	1.00

Graph Type: Heatmap (Correlation Matrix)

Description and Insights:
This heatmap illustrates the **Pearson correlation coefficients** between numerical features in the dataset, including the target variable churn. The correlation values range from **-1 (perfect negative)** to **+1 (perfect positive)**, with values closer to 0 indicating **no strong linear relationship**.

Key observations:

1. **Age** has the **highest positive correlation** with churn (0.29), suggesting that **older customers are more likely to churn**.

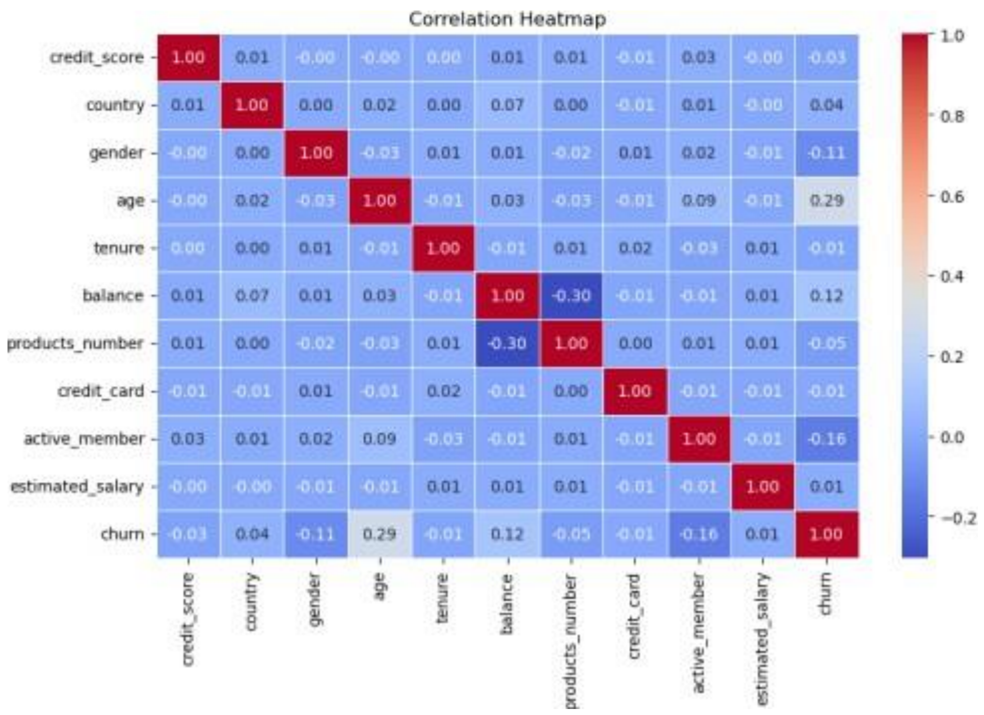
2. **Number of products** (products_number) has a **negative correlation** with churn (-0.30), indicating that customers with **fewer products are more prone to leave**, possibly due to low engagement.

3. Other variables like **balance (0.12)** and **credit score (-0.03)** have **weak correlations** with churn.

4. **Estimated salary** shows **almost no correlation** with churn, meaning it has **minimal impact** on customer retention in this context.

5. **Active membership** is **negatively correlated (-0.16)** with churn, indicating **active users are less likely to leave**.

Conclusion:
The features with the strongest influence on churn are **age**, **number of products**, and **active membership status**. These should be prioritized in feature selection and customer retention strategies. Variables with weak or no correlation may be less useful for churn prediction modeling.



Graph Type: Heatmap (Extended Correlation Matrix)

Description and Insights:
This heatmap presents the **correlation coefficients** between all numerical and encoded categorical variables in the dataset, including features like country and gender after label encoding.

Key findings:

1. **Age** again shows a **moderate positive correlation** with churn (0.29), indicating that **older customers are more likely to churn**.

2. **Products number** has a **negative correlation** with churn (-0.30), confirming that **customers using fewer products are more likely to leave**.

3. **Active member status** also has a **negative correlation** (-0.16) with churn, suggesting that **active customers are generally retained**.

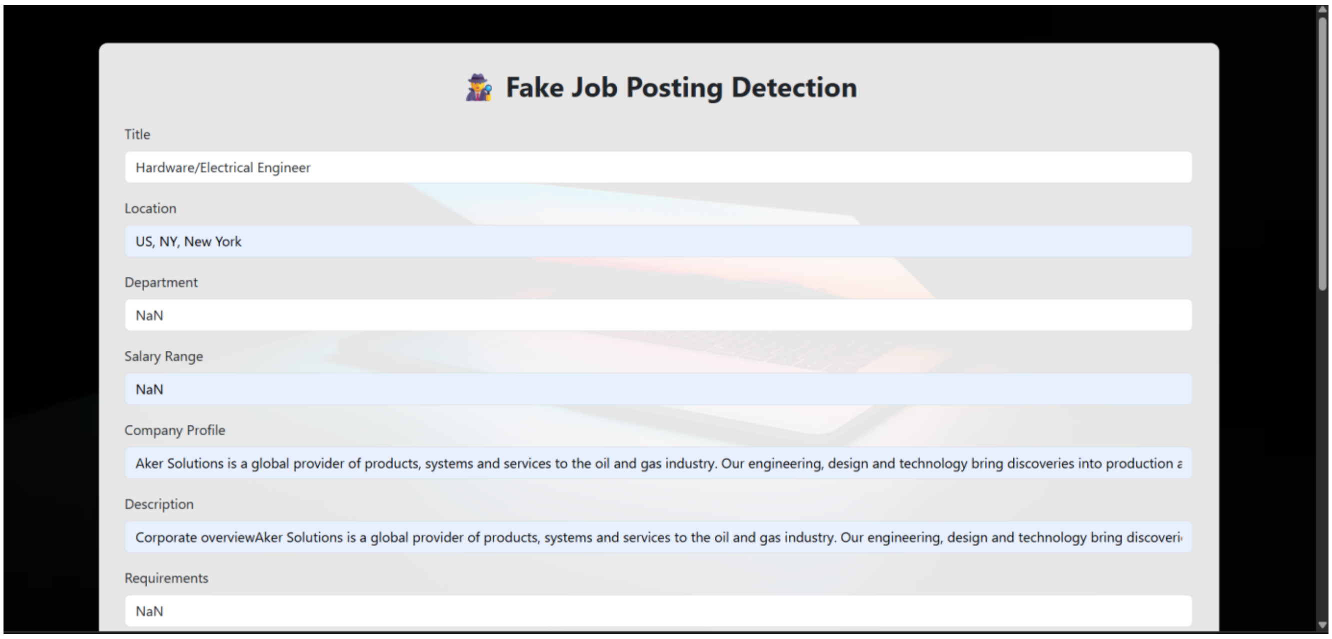
4. **Gender** shows a **mild negative correlation** with churn (-0.11), implying that **churn behavior may slightly vary by gender**, though this is relatively weak.

5. **Country** and **estimated salary** have **very low or negligible correlations** with churn, indicating that they **do not strongly influence customer retention** in this dataset.

6. **Credit card ownership**, **credit score**, and **tenure** also show minimal correlation with churn.

Conclusion:
The heatmap helps identify the most predictive features for the churn model. **Age**, **number of products**, and **active membership status** remain the most relevant factors, while features like **country**, **salary**, and **credit score** may have less predictive power in isolation.

- Screenshots:



🕵️ Fake Job Posting Detection

Title
Hardware/Electrical Engineer

Location
US, NY, New York

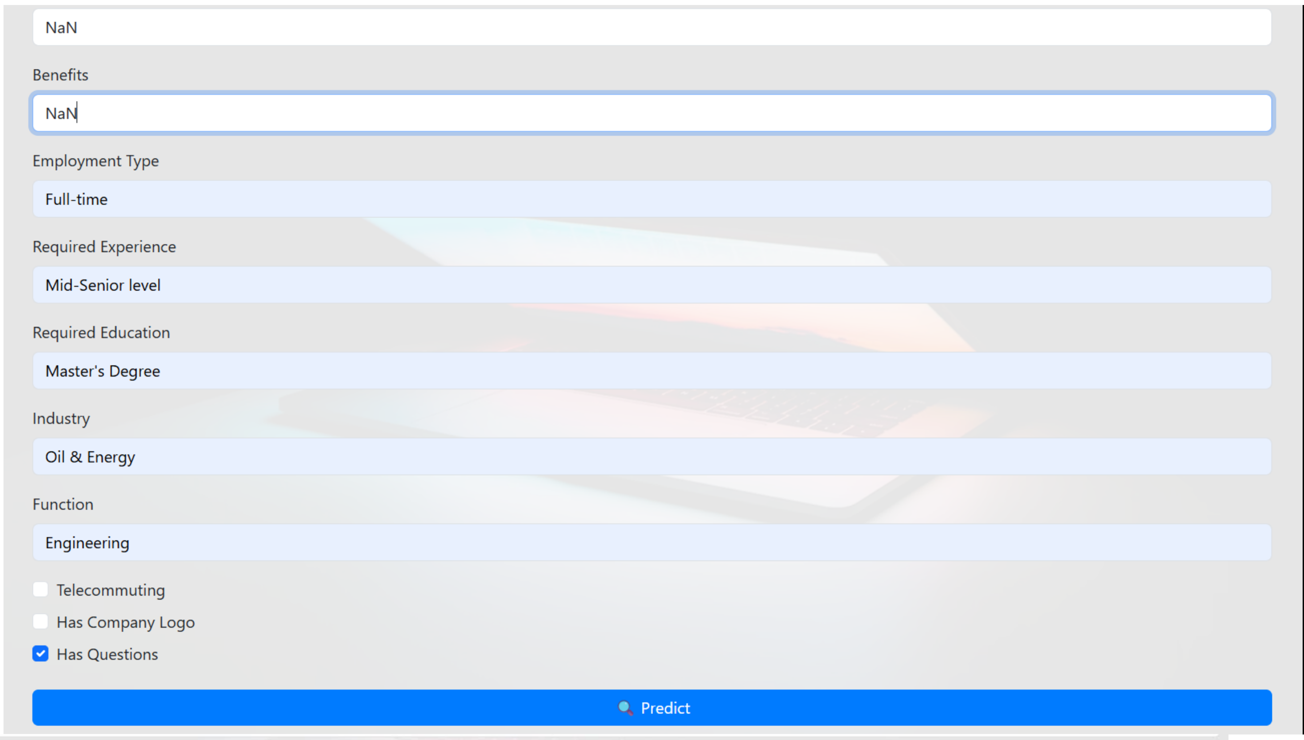
Department
NaN

Salary Range
NaN

Company Profile
Aker Solutions is a global provider of products, systems and services to the oil and gas industry. Our engineering, design and technology bring discoveries into production a

Description
Corporate overviewAker Solutions is a global provider of products, systems and services to the oil and gas industry. Our engineering, design and technology bring discoveri

Requirements
NaN



NaN

Benefits

NaN

Employment Type

Full-time

Required Experience

Mid-Senior level

Required Education

Master's Degree

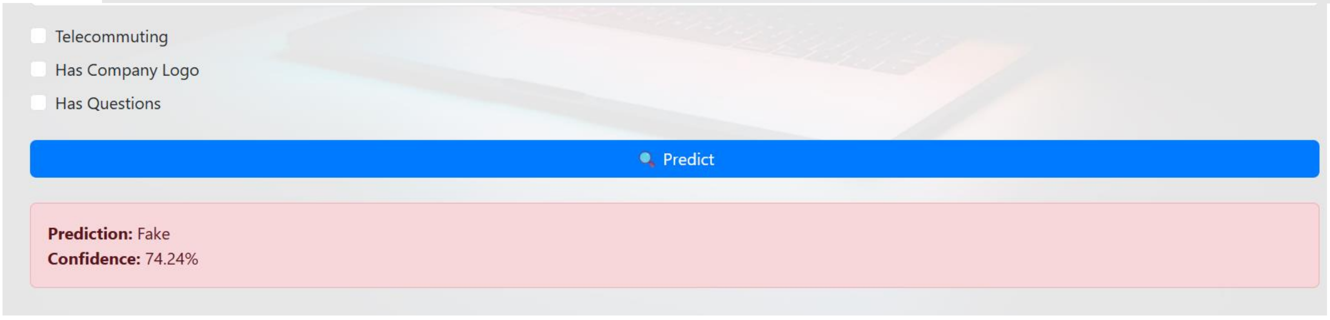
Industry

Oil & Energy

Function

Engineering

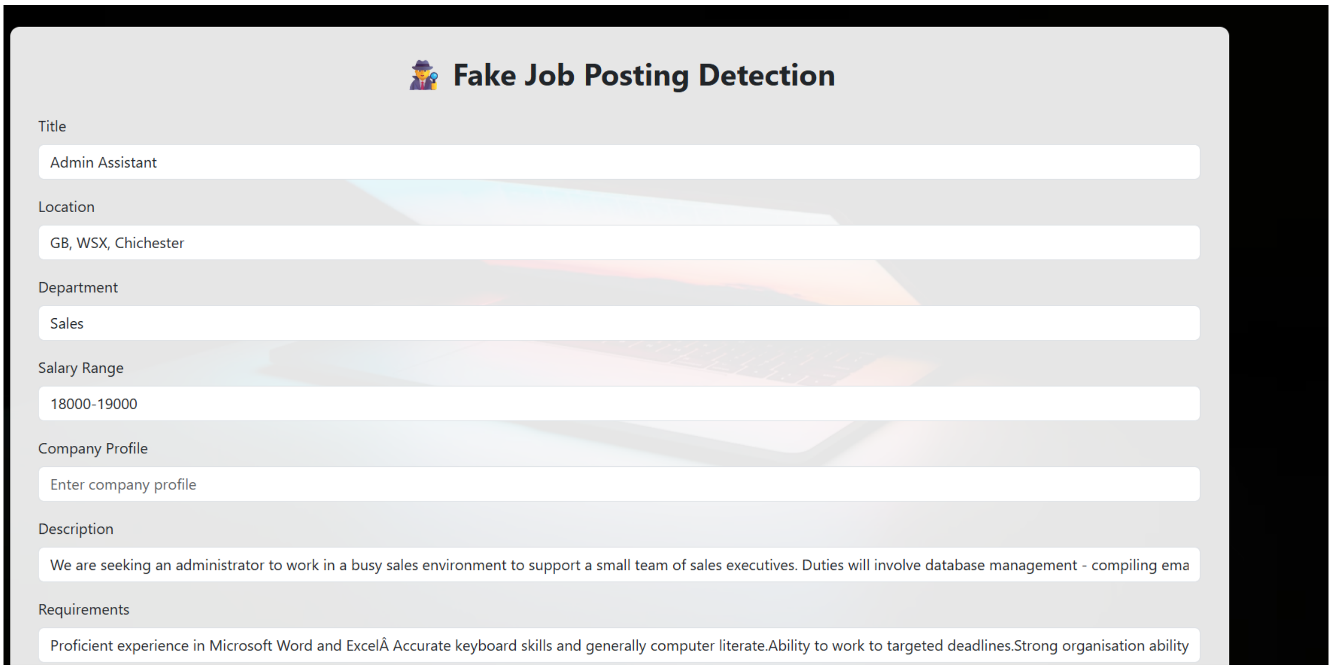
- [ ] Telecommuting
- [ ] Has Company Logo
- [x] Has Questions



- [ ] Telecommuting
- [ ] Has Company Logo
- [ ] Has Questions

Prediction: Fake
Confidence: 74.24%

Real:-



Fake Job Posting Detection

Title
Admin Assistant

Location
GB, WSX, Chichester

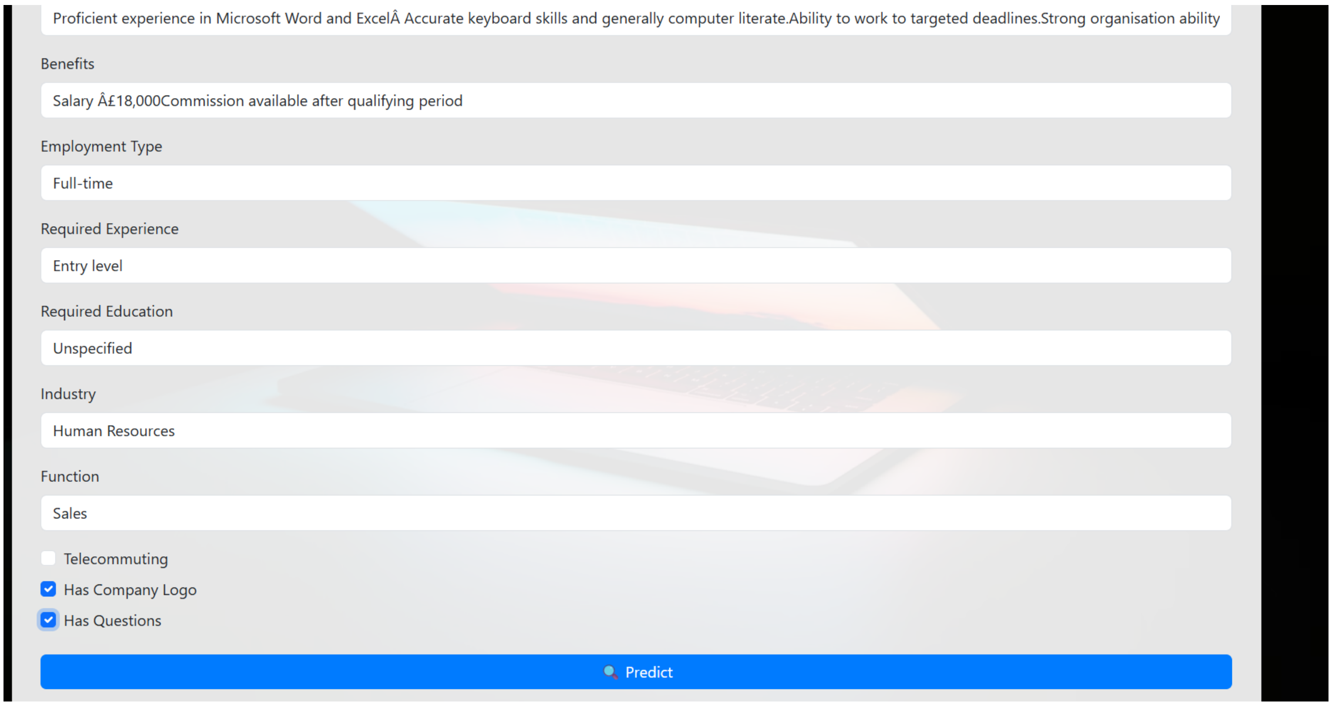
Department
Sales

Salary Range
18000-19000

Company Profile
Enter company profile

Description
We are seeking an administrator to work in a busy sales environment to support a small team of sales executives. Duties will involve database management - compiling ema

Requirements
Proficient experience in Microsoft Word and ExcelÂ Accurate keyboard skills and generally computer literate.Ability to work to targeted deadlines.Strong organisation ability



Proficient experience in Microsoft Word and ExcelÂ Accurate keyboard skills and generally computer literate.Ability to work to targeted deadlines.Strong organisation ability

Benefits
Salary Â£18,000Commission available after qualifying period

Employment Type
Full-time

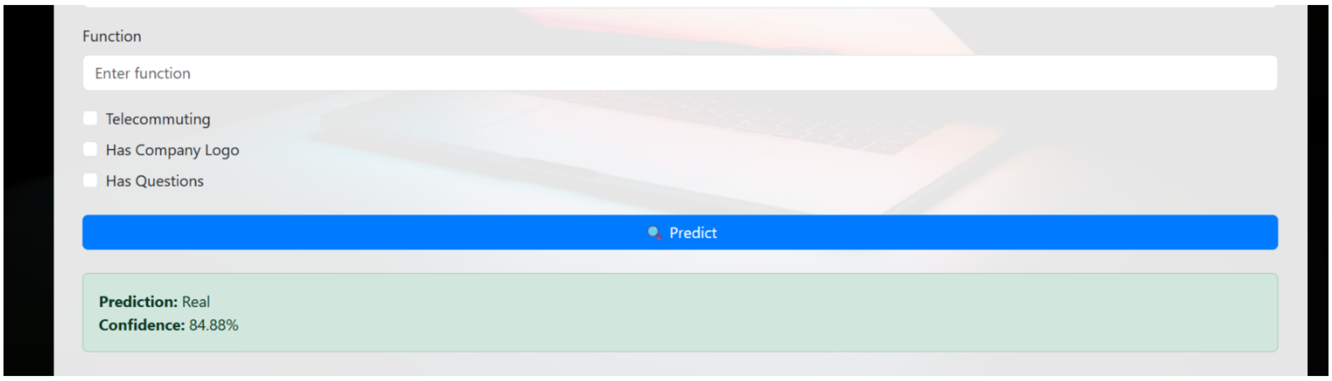
Required Experience
Entry level

Required Education
Unspecified

Industry
Human Resources

Function
Sales

☐ Telecommuting
☑ Has Company Logo
☑ Has Questions


Function

Enter function

Telecommuting

Has Company Logo

Has Questions

Prediction: Real
Confidence: 84.88%

Chapter 4: Results And Discussion

- **Output:** The core objective of this project, titled **Fake Job Posting Detection Using Machine Learning**, was to build a robust classification model that can distinguish between real and fake job listings. Using the cleaned and pre-processed fake_job_postings.csv dataset, several models were trained, evaluated, and compared. The following models were implemented:

  1. **Logistic Regression** – Used as a baseline classifier.
  2. **Random Forest Classifier (Default Settings)** – Offered more robust performance due to its ensemble structure.
  3. **Random Forest Classifier (Tuned)** – Hyperparameters were optimized using GridSearchCV to improve classification results.

  Before modeling, extensive data preprocessing was performed:

  - Dropping irrelevant columns like job_id
  - Combining textual features such as title, description, requirements, and benefits
  - Cleaning and transforming text (lowercasing, removing stopwords, lemmatization)
  - Vectorizing combined job text using TfidfVectorizer(max_features=500)
  - Encoding binary and categorical features (e.g., telecommuting, has_company_logo, employment_type)

Key Observations from Graphs:

1. **Extreme Class Imbalance:**
   - Only 5% of jobs were labeled as fake (fraudulent = 1).
   - Visualization via sns.countplot highlighted this stark imbalance, requiring precision-focused evaluation metrics.
2. **Word Cloud Patterns:**
   - Fake job posts included words like “money”, “apply now”, “online work”, and “quick”.

- Real job posts had more structured terms like “team”, “responsibilities”, and “requirements”.

3. **Correlation Insights:**
    - Features like has_company_logo and has_questions showed a slight inverse correlation with fake postings.
    - telecommuting and vague employment types were often flagged in fake jobs.

4. **Textual Distribution:**
    - Many fake jobs had missing or minimal descriptions and requirements.
    - They often lacked clear benefits or company profiles.

5. **Model Evaluation:**
    - The tuned Random Forest Classifier delivered the highest performance.
    - It achieved better F1-score, recall, and precision on fake job detection than baseline Logistic Regression.
    - Evaluation was based on classification_report and confusion_matrix, not just accuracy.

- **Challenges Faced:**
  Throughout the development of this machine learning pipeline, several technical and practical obstacles were encountered:
  1. **Extreme Class Imbalance**:
     - The dataset contained significantly fewer fake job postings.
     - This imbalance led to skewed accuracy if used as the sole metric.
     - Models were evaluated using **F1-score** and **confusion matrix** to balance precision and recall.

  2. **Inconsistent Textual Fields**:
     - Several rows had missing data in critical columns like company_profile, salary_range, and benefits.
     - These were either imputed with 'Unknown' or dropped based on threshold analysis.

  3. **Time-Consuming Grid Search**:
     - Hyperparameter tuning using GridSearchCV took several hours due to the size of the TF-IDF matrix (500 features).
     - To reduce complexity, cv=3 and a limited parameter grid were selected.

  4. **Unclear Label Definitions**:
     - In some rows, even real jobs had short or vague descriptions.
     - This added ambiguity in training and evaluation.

  5. **Deployment Compatibility**:
     - While Flask was sufficient locally, deploying the app to Render required configuring gunicorn, Procfile, and managing dependency issues via requirements.txt.

Learnings

This project offered substantial hands-on exposure across all stages of a full machine learning pipeline, from EDA to web deployment:

1. **Practical Data Preprocessing:**
   - Cleaning, merging, and encoding heterogeneous features.
   - Learned the importance of dealing with sparse and noisy real-world data.

2. **NLP and Vectorization:**
   - Used TF-IDF Vectorizer to transform job postings into numerical format.
   - Gained an understanding of why max_features selection (500) impacts both performance and speed.

3. **Model Development:**
   - Understood strengths and limitations of Logistic Regression vs. Random Forest for classification.

  - Ensemble methods like Random Forest provided better generalization on imbalanced data.

4. **Advanced Evaluation:**
    - Emphasized F1-score, confusion matrix, and ROC-AUC over traditional accuracy.
    - Realized the cost of false positives (misclassifying real jobs as fake) is lower than false negatives in this context.

5. **Flask-Based UI Integration:**
    - Built a user-friendly frontend using Flask, HTML/CSS, and Bootstrap.
    - Integrated model prediction with user form input, returning real-time results.

6. **Deployment Pipeline:**
    - Learned how to deploy apps on Render using GitHub integration.
    - Solved platform-specific issues such as missing dependencies, environment variables, and server startup commands (gunicorn app:app).

7. **Business Insight Extraction:**
    - Gained an appreciation for domain-specific signals such as the presence of a company logo or the specificity of job descriptions.
    - Developed an awareness of how machine learning can assist recruiters or job boards in flagging suspicious listings.

In conclusion, the Fake Job Posting Detection project was a holistic experience in applying machine learning to a real-world classification problem involving imbalanced textual data. It offered a blend of technical rigor and practical application, preparing the developer for advanced roles in data science and software deployment.

Chapter 5: Conclusion

The primary objective of this project was to design and implement a machine learning-based solution for the accurate identification of fake job postings using real-world data. With the exponential rise in online job scams, a reliable detection system can protect job seekers from fraudulent listings and preserve trust in recruitment platforms.

The project journey began with a comprehensive dataset (fake_job_postings.csv) comprising both structured and unstructured fields like job titles, descriptions, company profiles, and employment types. Through robust preprocessing—such as the removal of null values, merging of text fields, and TF-IDF vectorization of job descriptions—a clean and structured format was prepared for model training.

Exploratory Data Analysis (EDA) offered crucial insights into the dataset. It revealed that fake job postings often shared linguistic similarities: they used vague descriptions, lacked company logos, and avoided including applicant screening questions. Real jobs, on the other hand, were more detailed and well-structured. Word clouds, count plots, and correlation heatmaps were instrumental in shaping feature engineering decisions and revealed the significance of textual cues in classification.

Multiple models were trained, including Logistic Regression and Random Forest classifiers. The tuned Random Forest model outperformed others in precision, recall, and F1-score, making it the most suitable for deployment. Evaluation was conducted using classification metrics as well as visual tools like confusion matrices. Given the significant class imbalance, special attention was paid to recall and F1-score to minimize false negatives (real jobs misclassified as fake).

In addition to technical modeling, the project culminated in the development of a full-stack web application using Flask. This interface allows users to input job attributes and receive predictions on whether the listing is real or fake. The application was deployed on Render.com, demonstrating the project's real-world applicability.

In conclusion, this end-to-end machine learning pipeline provides a scalable and reliable method to detect fake job postings. It not only leverages advanced NLP techniques and robust classification models but also integrates them into an accessible web platform. The project also emphasized the importance of combining domain knowledge with technical expertise to design solutions that are both impactful and interpretable. With future iterations—including regular retraining and integration with job portals—this model can play a significant role in improving transparency and security in the job recruitment ecosystem.