Configuration Manual

MSc Research Project

APPLICATION OF NATURAL LANGUAGE PROCESSING FOR FAKE JOB POSTING DETECTION

Vaishnavi Gadikota

Student ID: 23393777

School of Computing

National College of Ireland

Supervisor: Arjun chikkankod

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: …Vaishnavi Gadikota…. ……

Student ID: …23393777……..……

Programme: MSc Data Analytics …… **Year:** …2025-2026…

Module: …Research practicum 2……

Lecturer: …Arjun chikkankod……

Submission Due Date: …11/12/2025…… ……..……

Project Title: APPLICATION OF NATURAL LANGUAGE PROCESSING FOR FAKE JOB POSTING DETECTION

……
……..……

Word Count:

…… **Page Count: 10**……

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……vaishnavi gadikota
……

Date: ……11/12/2025……
……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Vaishnavi Gadikota
Student ID: 23393777

1 Section 1

The importance of fake job posting detection has been growing as online recruitment websites receive more fraud adverts that take advantage of job seekers by using vague descriptions and lack of company information (Goud and Reddy, 2025). In this research, Natural Language Processing and machine learning are used to find a solution to this problem in the Kaggle dataset named Real or Fake Job Posting that has 17,880 listings (Bansal, 2020). The research takes the shape of a structured NLP pipeline including preprocessing, TF-IDF feature engineering and model evaluation. Four major models are used, including Logistic Regression, Naive Bayes, XGBoost, and LSTM, to compare the performance, detect linguistic features of a lie, and design a robust classification model to promote trust in online job markets.

2 Section 2

Hardware

- Storage capacity: 10GB-free (dataset, Python environment, model weights).
- CPU: Current multi-core processor (at least 4 cores are required).
- RAM: 16GB or 8GB is acceptable but 16GB is preferable.

Operating System

- Linux (Ubuntu), MacOS or Windows 10/11.
- The notebook will be authored in Google Colab and can be transported to any setting with an equal number of requirements.

3 Section 3

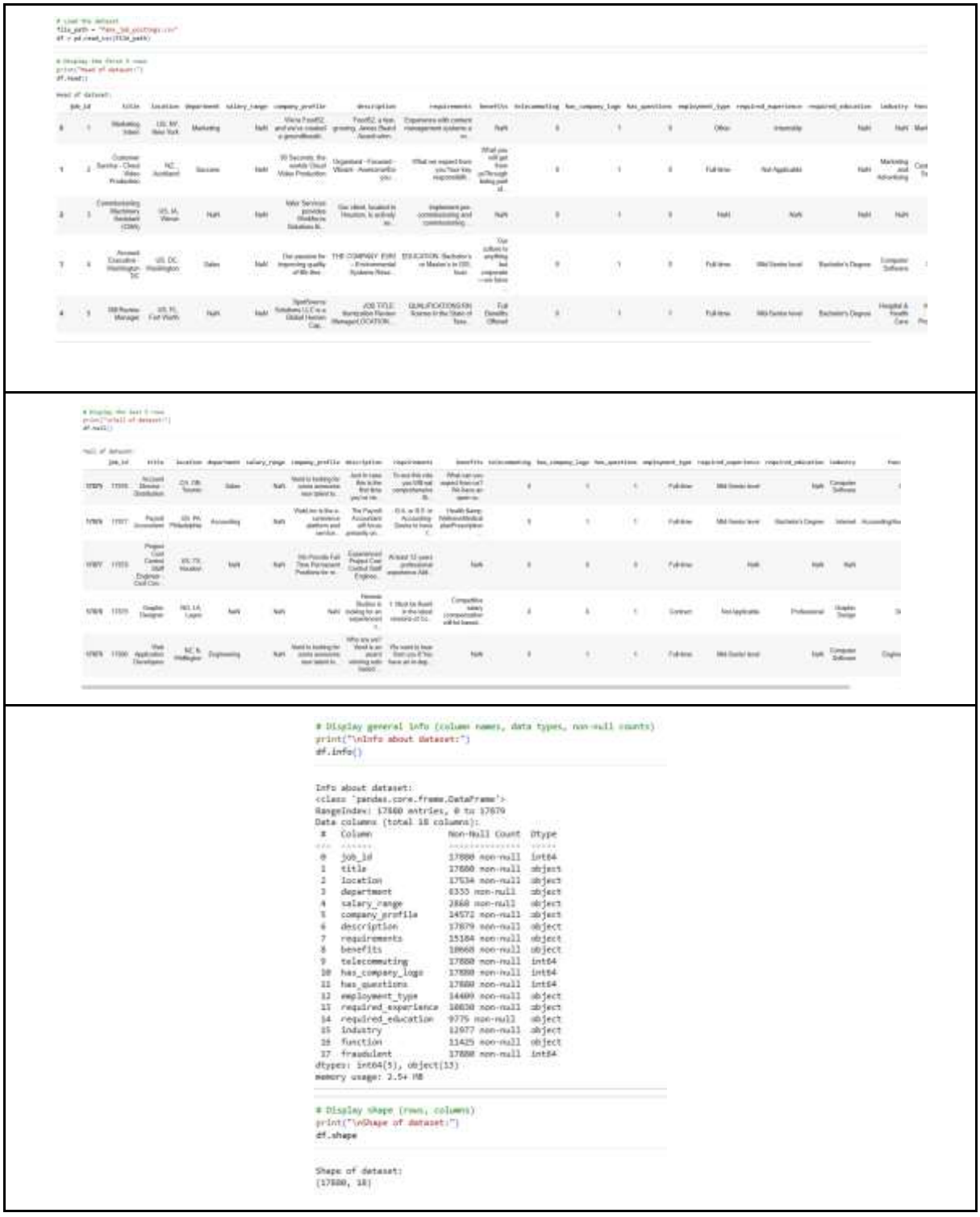
Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
nltk.download('punkt', quiet=True)
nltk.download('stopwords', quiet=True)
nltk.download('wordnet', quiet=True)
nltk.download('omw-1.4', quiet=True)
nltk.download('punkt_tab', quiet = True)
from wordcloud import WordCloud

from scipy.sparse import hstack, csr_matrix
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import ComplementNB
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score, roc_auc_score,
    classification_report, confusion_matrix
)
from joblib import dump
import json
import os
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.preprocessing import StandardScaler
from scipy import sparse
```

Figure 1: Importing key libraries

- File name: fake_job_postings.csv (50.06 MB)
- Kaggle (2020). *Real / Fake Job Posting Prediction*. [online] Kaggle.com. Available at: https://www.kaggle.com/datasets/shivamb/real-or-fake-fake-jobposting-prediction/data
- Separator: Comma-separated values (CSV).

Data loading and checking



```
Info about dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17880 entries, 0 to 17879
Data columns (total 18 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   job_id               17880 non-null  int64
 1   title                17880 non-null  object
 2   location             17534 non-null  object
 3   department           6333 non-null   object
 4   salary_range         2868 non-null   object
 5   company_profile      14572 non-null  object
 6   description          17879 non-null  object
 7   requirements         15184 non-null  object
 8   benefits             10668 non-null  object
 9   telecommuting        17880 non-null  int64
 10  has_company_logo     17880 non-null  int64
 11  has_questions        17880 non-null  int64
 12  employment_type      14409 non-null  object
 13  required_experience  10830 non-null  object
 14  required_education   9775 non-null   object
 15  industry             12977 non-null  object
 16  function             11425 non-null  object
 17  fraudulent           17880 non-null  int64
dtypes: int64(5), object(13)
memory usage: 2.5+ MB
```

```
Shape of dataset:
(17880, 18)
```

Figure 2: Loading and checking of the dataset

- Displayed the first and last five rows of the data to graphically examine the data structure, missing data, and the column format.
- Examined dataset statistics with df.info(), which determines the types of data, non-NULLs and the total number of entries in all columns.

- Removed the job -id column since it was not a predictor to model training but it was just an identifier.

Data Quality checking

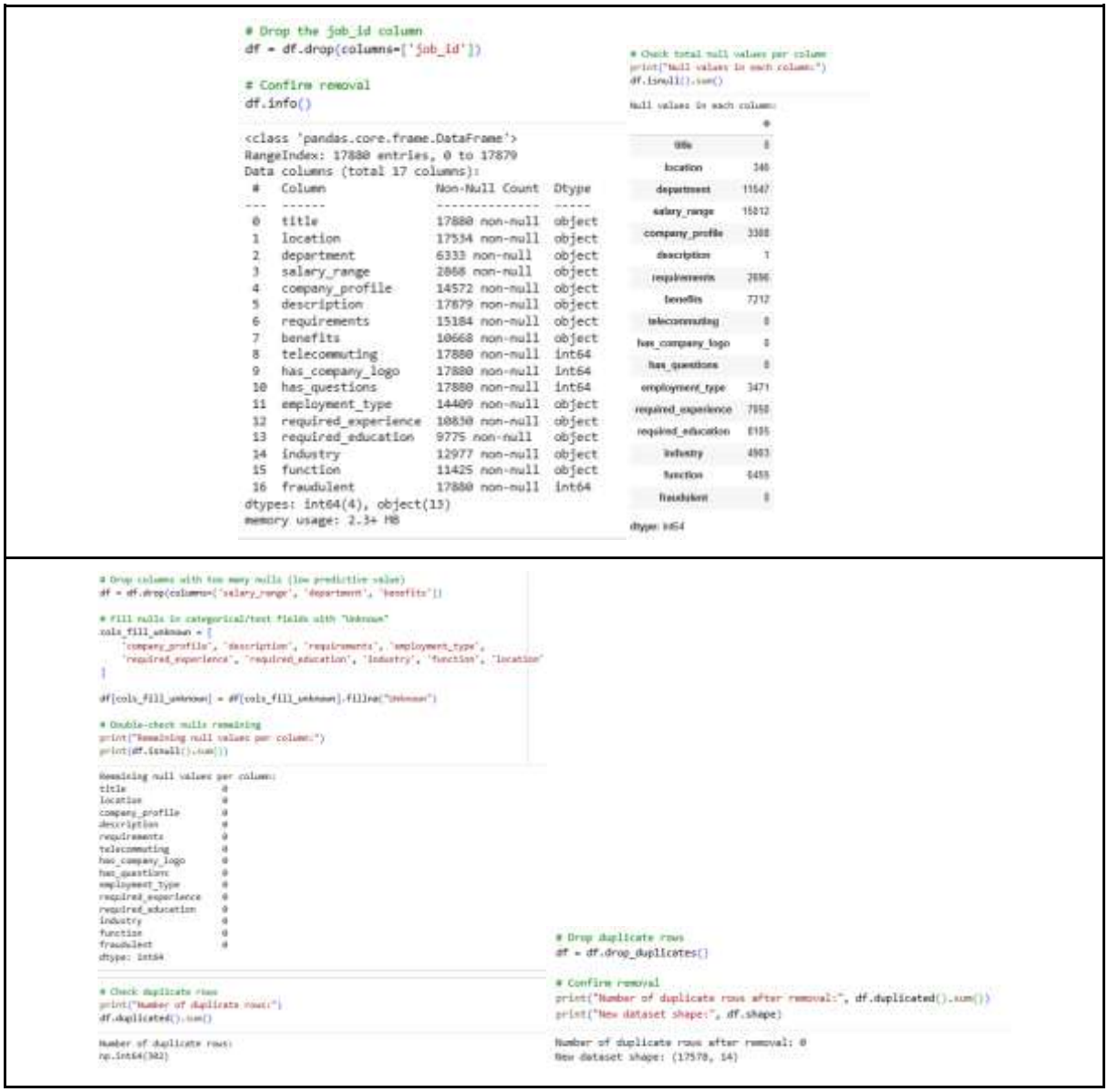


Figure 3: Data Quality checking

- Computed the total number of null values per column to find out which features needed to be cleaned or deleted.
- Dropped columns salary_range, department, and benefits, because they contained extremely high missing values and contributed little to prediction.
- Replaced missing values in categorical columns such as company_profile, description, requirements, employment_type, required_experience, required_education, industry, function, and location with the placeholder "Unknown".
- Retested values that were absent to ensure that all the null values were filled or deleted.

- Uniqueness of rows was ensured by dropping duplicate rows so that there is no model bias due to job adverts being repeated.
- Final dataset upon cleaning, with the row and column numbers updated to indicate successful preprocessing.

Text based preprocessing using NLP

```
# Clean + tokenize + remove stopwords + lemmatize (per document)
STOPWORDS = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def normalize_and_clean(text: str) -> str:
    """
    Normalize to lowercase; remove HTML, URLs, numbers, punctuation; collapse spaces.
    """
    text = text.lower().strip()
    text = re.sub(r'<[^>]+>', ' ', text)              # remove HTML tags
    text = re.sub(r'http\S+|www\.\S+', ' ', text)     # remove URLs
    text = re.sub(r'[^a-z\s]', ' ', text)             # keep only letters and spaces
    text = re.sub(r'\s+', ' ', text).strip()          # normalize whitespace
    return text

def tokenize_remove_stop_lemma(text: str) -> list:
    """
    Tokenize + remove stopwords + lemmatize.
    """
    tokens = nltk.word_tokenize(text)                 # tokenization
    tokens = [t for t in tokens if t not in STOPWORDS]  # stopword removal
    tokens = [lemmatizer.lemmatize(t) for t in tokens]  # lemmatization
    return tokens

def preprocess_doc(text: str) -> list:
    """Apply the above steps and return token list."""
    text = normalize_and_clean(str(text))
    tokens = tokenize_remove_stop_lemma(text)
    return tokens
```

```
# Handle rare words (corpus-level filtering)
from collections import Counter

def build_vocab_and_filter(token_lists, min_freq: int = 5):
    """
    Build frequency dictionary over the whole corpus and drop tokens with freq < min_freq.
    Returns filtered token lists and the frequency Counter.
    """
    freq = Counter()
    for toks in token_lists:
        freq.update(toks)
    # keep only tokens meeting minimum frequency
    filtered = [[t for t in toks if freq[t] >= min_freq] for toks in token_lists]
    return filtered, freq
```

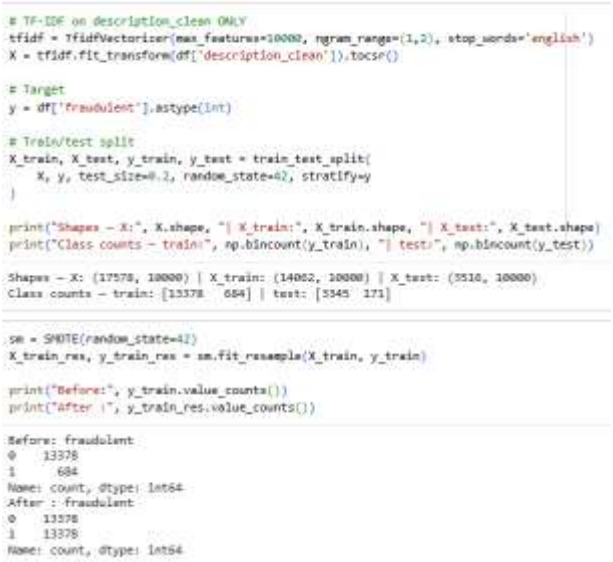
```
# Text columns to preprocess
text_cols = ['title', 'company_profile', 'description', 'requirements']

# Ensure no NaNs in these columns before processing (optional but practical)
df[text_cols] = df[text_cols].fillna("")
```

```
# Text columns to preprocess
text_cols = ['title', 'company_profile', 'description', 'requirements']

# Ensure no NaNs in these columns before processing (optional but practical)
df[text_cols] = df[text_cols].fillna("")

# Per-column tokenization pipeline
tokenized_data = {}
for col in text_cols:
    tokenized_data[col] = df[col].apply(preprocess_doc).tolist()

# Rare-word filtering per column (choose a sensible threshold; 5 is common)
MIN_FREQ = 5
filtered_tokenized = {}
vocab_stats = {}

for col in text_cols:
    filtered, freq = build_vocab_and_filter(tokenized_data[col], min_freq=MIN_FREQ)
    filtered_tokenized[col] = filtered
    vocab_stats[col] = freq  # Counter of token frequencies for inspection if needed

# Join tokens back to strings for downstream vectorizers (e.g., TF-IDF)
for col in text_cols:
    df[col + "_clean"] = [" ".join(toks) for toks in filtered_tokenized[col]]

# Pick the text columns
text_cols = ['title', 'company_profile', 'description', 'requirements']

# Show original vs. cleaned version for first few rows
for col in text_cols:
    print(f"\n--- {col} (Original vs. Cleaned) ---")
    print("Original:")
    print(df[col].head(2).values)  # original text
    print("\nCleaned:")
    print(df[col + "_clean"].head(2).values)  # after preprocessing

# Check if any of the cleaned columns still have NaN
print("\nNaN values in cleaned text columns:")
print(df[[col + "_clean" for col in text_cols]].isnull().sum())
```

Figure 4: Text based preprocessing using NLP

- Implemented NLP cleaning in text_preprocessing.py using functions normalize_and_clean(), tokenize_remove_stop_lemm(), and preprocess_doc().
- Removed HTML, URLs, punctuation, stopwords, and applied lemmatization.
- Used build_vocab_and_filter() for rare-word filtering (min_freq=5).
- Processed text columns (title, company_profile, description, requirements) and generated cleaned versions for model input.

Feature engineering, data splitting and balancing



```
# TF-IDF on description_clean ONLY
tfidf = TfidfVectorizer(max_features=10000, ngram_range=(1,2), stop_words='english')
X = tfidf.fit_transform(df['description_clean']).tocsr()

# Target
y = df['fraudulent'].astype(int)

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

print("Shapes — X:", X.shape, "| X_train:", X_train.shape, "| X_test:", X_test.shape)
print("Class counts — train:", np.bincount(y_train), "| test:", np.bincount(y_test))

Shapes — X: (17578, 10000) | X_train: (14062, 10000) | X_test: (3516, 10000)
Class counts — train: [13378   684] | test: [3345  171]

sm = SMOTE(random_state=42)
X_train_res, y_train_res = sm.fit_resample(X_train, y_train)

print("Before:", y_train.value_counts())
print("After :", y_train_res.value_counts())

Before: fraudulent
0    13378
1      684
Name: count, dtype: int64
After : fraudulent
0    13378
1    13378
Name: count, dtype: int64
```

Figure 5: Feature engineering, data splitting and balancing

- In feature engineering.py, use Tfidfvectorizer() on description clean.
- Split dataset using train_test_split().
- Balanced classes with SMOTE using sm.fit_resample(), with an equal sample of fraud and non-fraud to model.

Implementation of Models

XGBoost classifier

```
# Train XGBoost on the resampled training set
xgb_clf = XGBClassifier(
    n_estimators=300,
    max_depth=6,
    learning_rate=0.1,
    subsample=0.9,
    colsample_bytree=0.9,
    random_state=42,
    n_jobs=-1,
    eval_metric="logloss"  # avoids warning
)
xgb_clf.fit(X_train_res, y_train_res)
```


Figure 6: Performance of XGBoost

Logistic Regression

```
# Train Logistic Regression on the resampled training set
log_clf = LogisticRegression(
    solver='liblinear',
    max_iter=1000,
    random_state=42
)
log_clf.fit(X_train_res, y_train_res)
```

```
Accuracy: 95.82%
Precision (Fake=1): 49.27%
Recall (Fake=1): 78.95%
F1-score (Fake=1): 60.67%
ROC-AUC: 96.85%

Classification Report (%):
Class 0 -> Precision: 98.88% | Recall: 96.84% | F1: 97.84% | Support: 3345
Class 1 -> Precision: 49.27% | Recall: 78.95% | F1: 60.67% | Support: 171
Macro Avg -> Precision: 74.08% | Recall: 87.40% | F1: 79.01%
Weighted Avg -> Precision: 96.48% | Recall: 95.82% | F1: 96.56%
Overall Accuracy: 95.82%
```

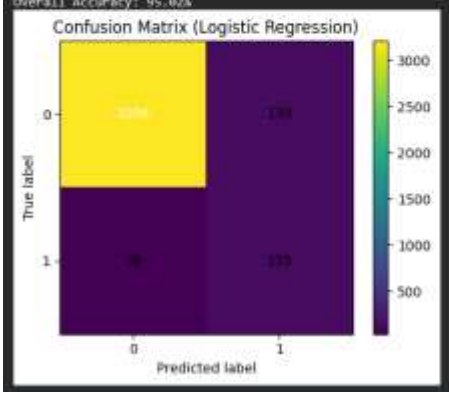


Figure 7: Performance of Logistic Regression

Naive Bayes

```
# Naive Bayes

# Train Naive Bayes on the resampled training set
nb_clf = ComplementNB()
nb_clf.fit(X_train_res, y_train_res)
```

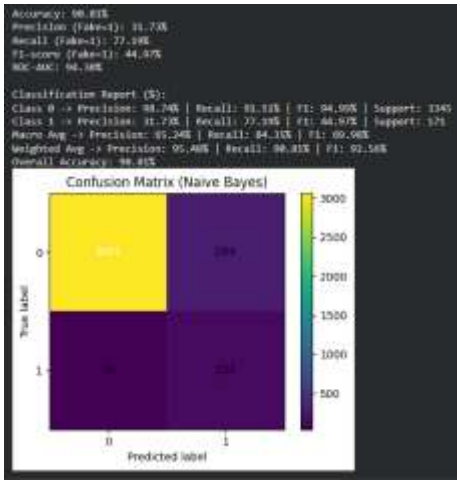


Figure 8: Performance of Naive Bayes

LSTM

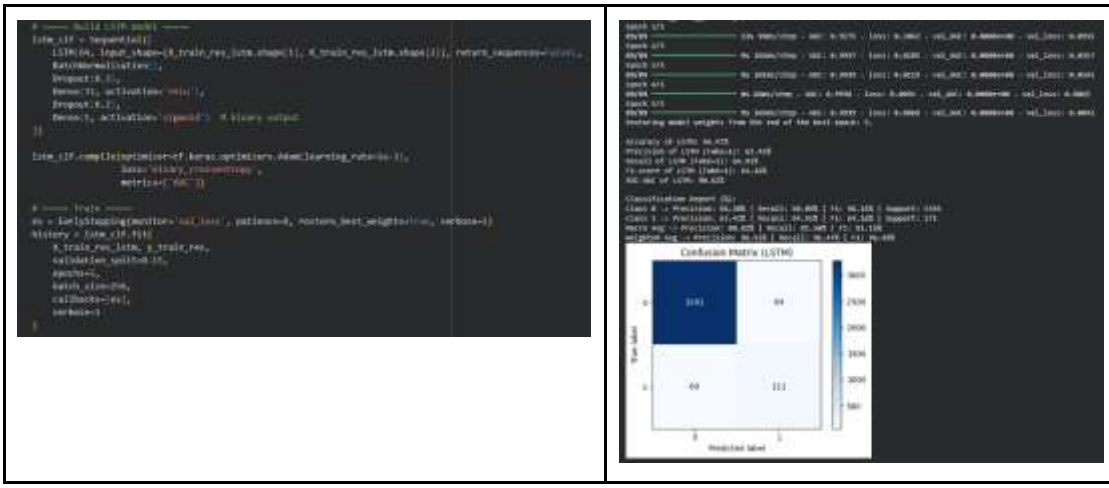
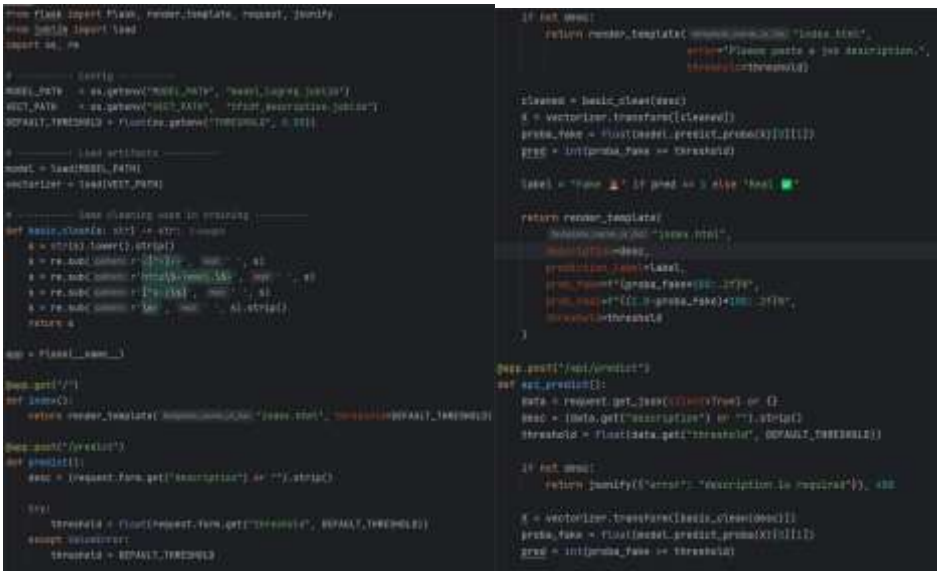


Figure 9: Performance of LSTM

This has been same for all the machine learning models from figure 6 to 9:

- Formulation of each classification model resulted in performance based on TF-IDF or sequence input and achieved performance metrics using the Scikit-learn Classification_report() and Metrics functions: Accuracy, Precision, Recall, F1-score and ROC-AUC scores.
- To visualise how well each of the classification models performed in terms of correct versus incorrect predictions, confusion matrices were created for each model using confusion_matrix() from Scikit-learn and visualised using Matplotlib.
- Comparative performance of the different classification models was tabulated along with a final ranking order based on Recall, Precision, F1-score, ROC-AUC.

Flask implementation for real-world fake job deployment



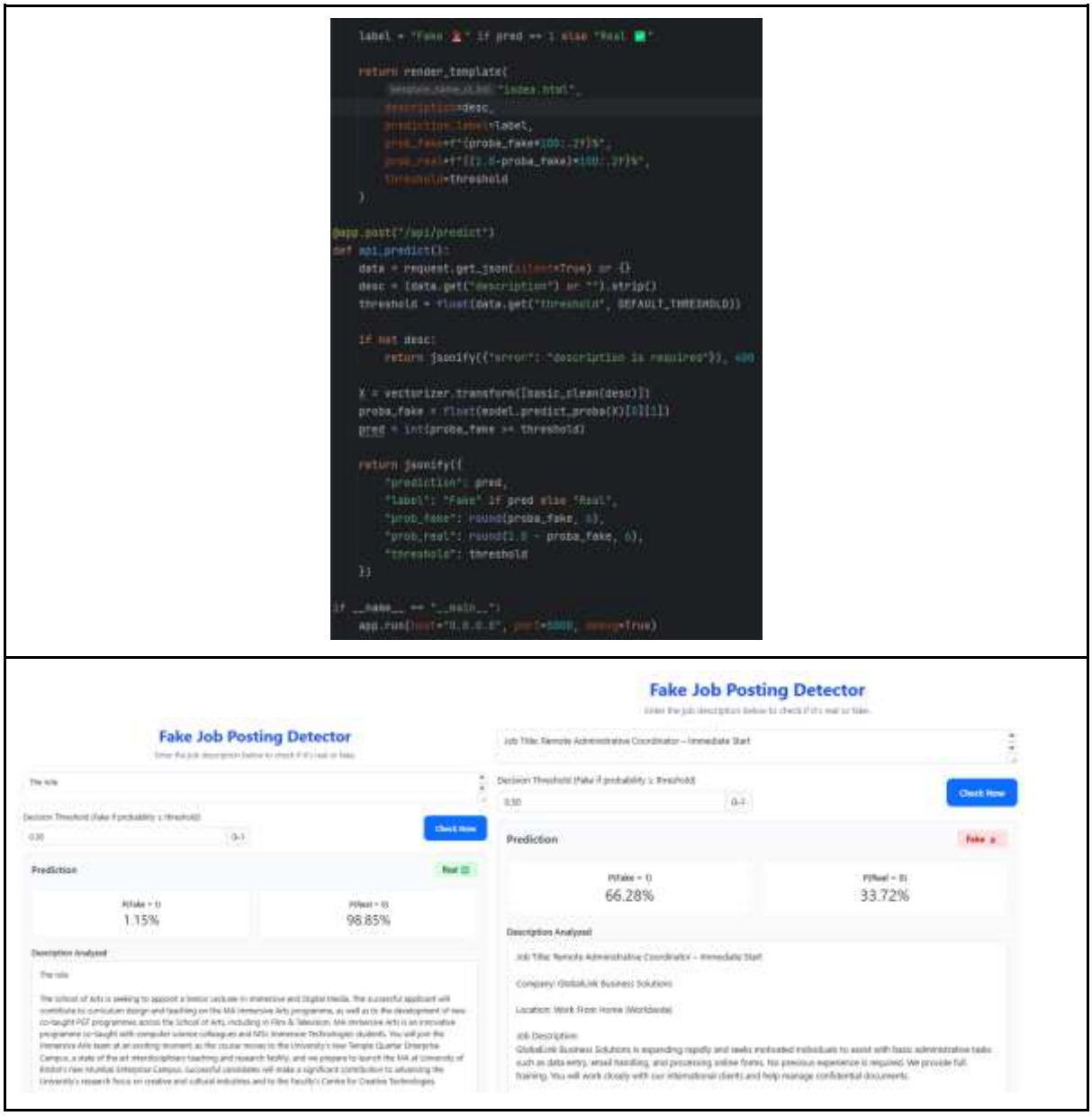


Figure 10: Flask-based web application

- Created using Flask, a Python micro framework, to deploy the application on the web.
- Using pickle/joblib, the model and preprocessing pipeline were saved and were imported in the web application for real-time predictions.
- Designed using HTML, CSS and Bootstrap in the main folder of Flask. Backend prediction interface is served using the Flask routing decorator @app.route('/', methods=['GET','POST']).
- Predicted output (Fake/Real) is shown along with its probability on the application's front-end interface.
- Dynamic threshold adjustment has been added to the application through user supplied HTML form input.
- Locally hosted using the development server of Flask for testing of its actual use to detect fake and real websites.

References

Bansal, S. (2020). *Real / Fake Job Posting Prediction*. [online] Kaggle.com. Available at: https://www.kaggle.com/datasets/shivamb/real-or-fake-fake-jobposting-prediction/data [Accessed 10 Dec. 2025].

Goud, T.N. and Reddy, R. (2025). A Machine Learning Approach for Detecting Fraudulent Job Postings in Online Recruitment Platforms. *Journal of Sensors, IoT & Health Sciences*, 3(3), pp.14–28. doi:https://doi.org/10.69996/jsihs.2025012.