Configuration Manual

MSc Research Project
Msc Data Analytics

Aishwarya Dinesh Rathudi
Student ID: X21222762

School of Computing
National College of Ireland

Supervisor: Abubakr Siddig

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Aishwarya Dinesh Rathudi ……. ………………………………………………………………………………
Student ID:	x21222762 ……………………………………………………………………………………….……
Programme:	Msc Data Analytics ……………………………………………………… **Year:** 2022-2023 …………………………..
Module:	Msc Research Project ……………………………………………………………………………….………
Lecturer:	Abubakr Siddig ……………………………………………………………………………….………
Submission Due Date:	18-09-2023 ……………………………………………………………………………….………
Project Title:	Fake Job Post Prediction ……………………………………………………………………………….………
Word Count:	768 …………………………………… **Page Count:** 19 …………………………….…….………

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: Aishwarya Dinesh Rathudi ………………………………………………………………………………………………

Date: 18-09-2023 ………………………………………………………………………………………………

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

Aishwarya Dinesh Rathudi
Student ID: X21222762

1 Introduction

The setup, configuration, and efficient use of a system, piece of software, piece of hardware, tools or procedure are all covered in a configuration handbook. It seeks to give detailed information about the Project. It provides a wealth of details at each stage of the project journey and serves as a guiding light that illuminates the way towards efficient use.

2 System Requirements

- **Hardware Requirements**

Device specifications — Copy

Device name	LAPTOP-JJ5VV3K1	
Processor	AMD Ryzen 5 4500U with Radeon Graphics	2.38 GHz
Installed RAM	8.00 GB (7.42 GB usable)	
Device ID	02B68062-BE8E-4A5C-B333-3A66A3360C2E	
Product ID	00327-35897-79379-AAOEM	
System type	64-bit operating system, x64-based processor	
Pen and touch	No pen or touch input is available for this display	

Fig 1. Hardware Specifications

- **Software Requirements**

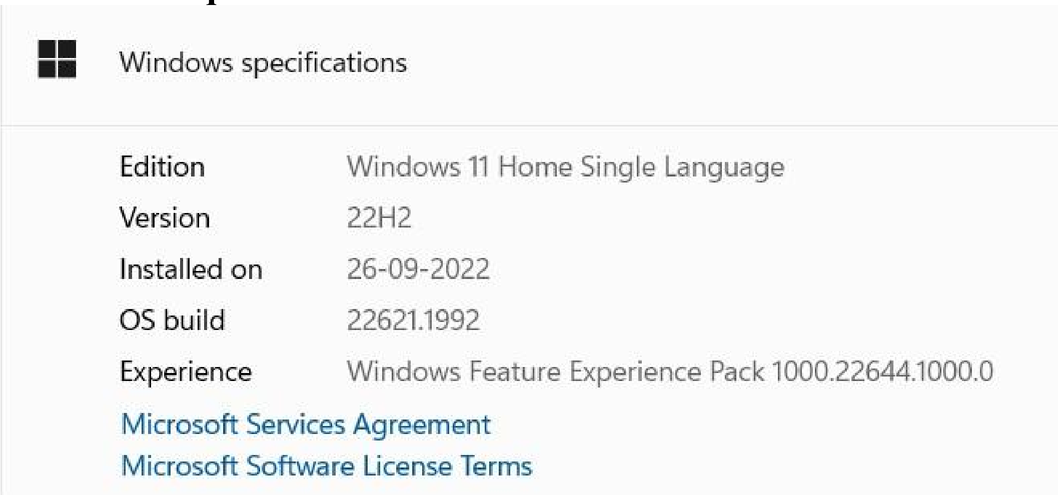

Fig 2. Software Specifications

- **Google Colab**

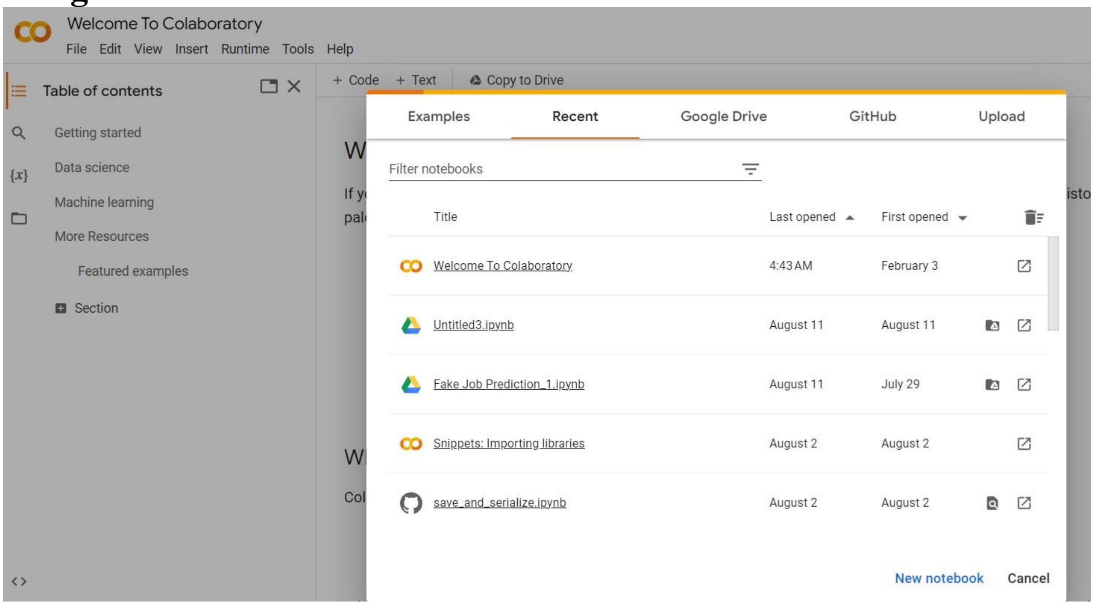

Fig 3. Google Colab

3 Dataset

The Dataset is collected from Kaggle, it contains 17880 rows and 18 columns.

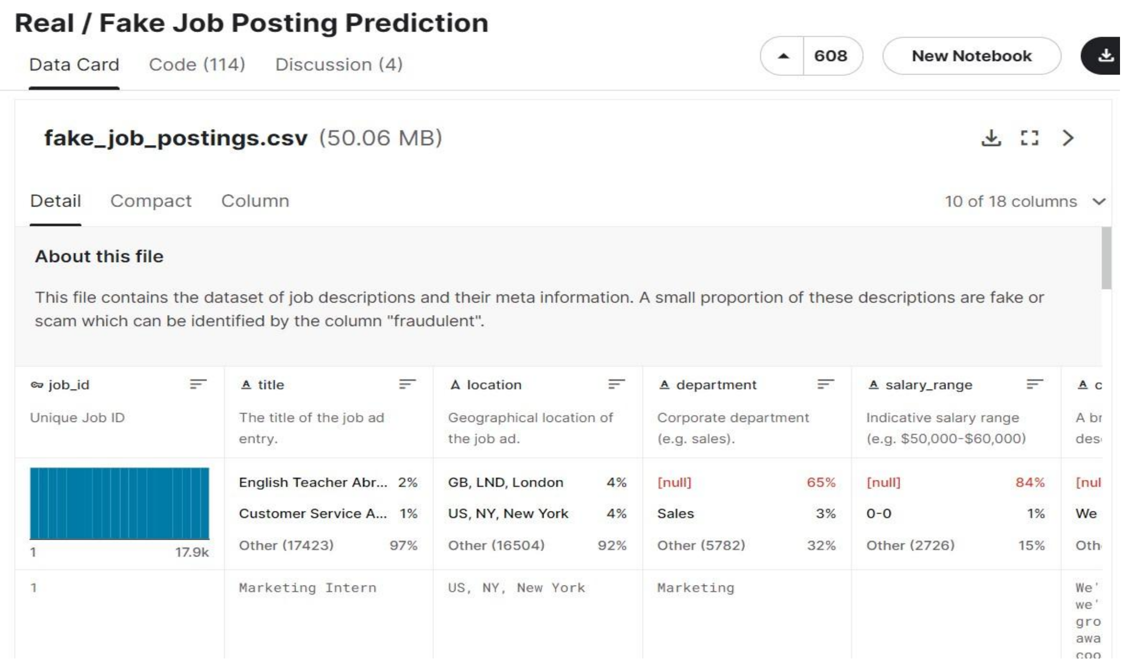

Fig 4. Data

4 Dataset Loading

```
[ ] # import the libraries
    import numpy as np
    import pandas as pd
    import seaborn as sb
    import matplotlib.pyplot as plt
    from sklearn.model_selection import train_test_split
    from sklearn.base import TransformerMixin
    from sklearn.utils import resample
    from gensim.models import Word2Vec
    from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D, Dense, Dropout, Flatten, MaxPooling1D, LSTM
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.preprocessing.text import Tokenizer
    from tensorflow.keras.preprocessing.sequence import pad_sequences
    from tensorflow.keras.optimizers import Adam
    from tensorflow.keras.callbacks import EarlyStopping
    from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
```

```
[ ] import csv
    df=pd.read_csv("fake_job_postings.csv")
    df.head(5)
```

Fig 5. Load Data

5 Exploratory Data Analysis (EDA)

The above figure shows the summary of the dataset.

```
df.info()
```

```
<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         15185 non-null  object
 8   benefits             10670 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
```

Fig 6. Summary of data

Here, Splitting the location into country, so that country-wise job posting can be visualize.

```
df['country']=df['location'].apply(lambda x:x.split(',')[0])
```

Fig 7. Split_location

Visualize job posting country-wise

```
#visualize country-wise job posting
country =dict(df.country.value_counts()[:11])

del country[' ']
plt.figure(figsize=(8,4))

plt.title('Country-wise Job Posting',size=20)
plt.bar(country.keys(),country.values())

plt.xlabel('Name of Country')
plt.ylabel('Number of Jobs')
```

Fig 8. Plot Country-wise job posting

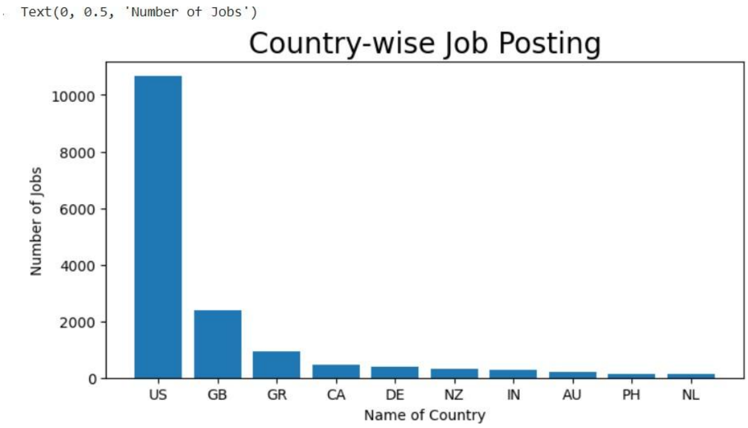

Fig 9. country-wise job posting

Viualize the Experience-wise job posting

```
# Visualize Job posting by Experience

experience =dict(df.required_experience.value_counts()[:11])

del experience[' ']
plt.figure(figsize=(8,4))

plt.title('Experience-wise Job Posting',size=20)
plt.bar(experience.keys(),experience.values())

plt.xlabel('Experience')
plt.ylabel('Number of Jobs')
```

Fig 10. Plot Experience-wise job posting

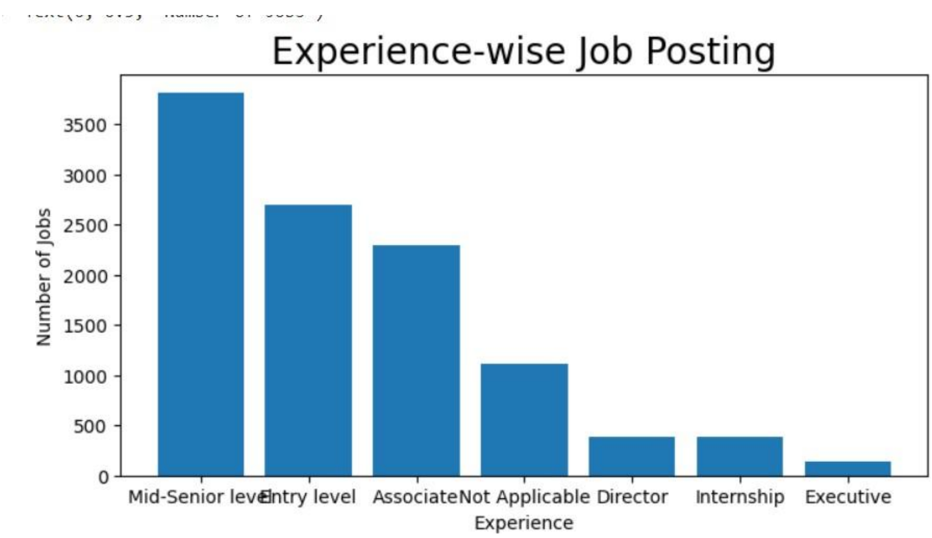

Fig 11. Experience-wise job posting

Count of jobs title which are frequent

```
# title of jobs which are frequent.
print(df.title.value_counts()[:10])
```

```
English Teacher Abroad                                 311
Customer Service Associate                             146
Graduates: English Teacher Abroad (Conversational)     144
English Teacher Abroad                                  95
Software Engineer                                       86
English Teacher Abroad (Conversational)                 83
Customer Service Associate - Part Time                  76
Account Manager                                         75
Web Developer                                           66
Project Manager                                         62
Name: title, dtype: int64
```

Fig 12. Title of jobs

```
[ ] #Fake job titles
    df[df.fraudulent == 1].title.value_counts()[:11]
```

```
Data Entry Admin/Clerical Positions - Work From Home                      21
Home Based Payroll Typist/Data Entry Clerks Positions Available           21
Cruise Staff Wanted *URGENT*                                              21
Customer Service Representative                                           17
Administrative Assistant                                                  16
Home Based Payroll Data Entry Clerk Position - Earn $100-$200 Daily       12
Account Sales Managers $80-$130,000/yr                                    10
Network Marketing                                                         10
Payroll Clerk                                                             10
Payroll Data Coordinator Positions - Earn $100-$200 Daily                 10
Data Entry                                                                 9
Name: title, dtype: int64
```

Fig 13. Fake job titles

```
#Non-fraud job titles
df[df.fraudulent == 0].title.value_counts()[:11]
```

```
English Teacher Abroad                                 311
Customer Service Associate                             146
Graduates: English Teacher Abroad (Conversational)     144
English Teacher Abroad                                  95
Software Engineer                                       86
English Teacher Abroad (Conversational)                 83
Customer Service Associate - Part Time                  76
Account Manager                                         73
Web Developer                                           66
Project Manager                                         62
Beauty & Fragrance consultants needed                   60
Name: title, dtype: int64
```

Fig 14. Non-fraud job titles

Wordcloud is used to generate the frequency of word

```
# Check the frequency of Word in datasets

from wordcloud import WordCloud
all_words = ' '.join([text for text in df['text']])
wordcloud = WordCloud(width = 600 ,height = 300, max_font_size = 120).generate(all_words)

plt.figure(figsize =(8,4))
plt.imshow(wordcloud)
plt.show()
```

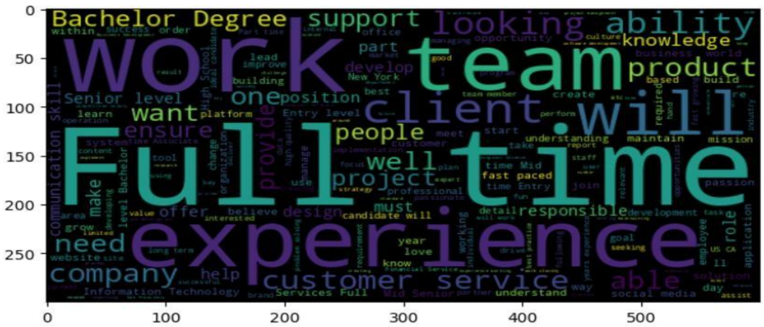

Fig 15. Frequency of word

```
# frequency of word in real posting of jobs
real_post = ' '.join([text for text in df['text'][df['fraudulent']==0]])
wordcloud1 = WordCloud(width = 600 ,height = 300, max_font_size = 120).generate(real_po
plt.figure(figsize =(8,4))
plt.imshow(wordcloud1)
plt.show()
```

Fig 16. Frequency of word in real jobs posting

```
# frequency of word in fake posting of jobs
fake_post = ' '.join([text for text in df['text'][df['fraudulent'] == 1]])
wordcloud2 = WordCloud(width = 600 ,height = 300, max_font_size = 120).generate(fake_post)

plt.figure(figsize =(8,4))
plt.imshow(wordcloud2)
plt.show()
```

Fig 17. Frequency of word in fake jobs posting

6 Data Preprocessing

```
[ ] #check for null values
    df.isna().sum()

    job_id                     0
    title                      0
    location                 346
    department             11547
    salary_range           15012
    company_profile         3308
    description                1
    requirements            2695
    benefits                7210
    telecommuting              0
    has_company_logo           0
    has_questions              0
    employment_type         3471
    required_experience     7050
    required_education      8105
    industry                4903
    function                6455
    fraudulent                 0
    dtype: int64
```

Fig 18. Null values

```
[ ] #filling ''
    df.fillna(' ',inplace=True)
```

Fig 19. Filling null values

```
[ ] #Removing the unwanted columns
    df.drop(columns=['job_id','salary_range','department','telecommuting','has_company_logo','has_questions'],inplace=True)
```

Fig 20. Drop columns

```
[ ] df['fraudulent'].value_counts().to_frame()
```

	fraudulent
0	17014
1	866

Fig 21. Count of fraudulent vs Non-fraudulent

```
[ ]  #text preprocessing
     import re
     import nltk
     nltk.download('stopwords')
```

Fig 22. Import libraries

```
[ ] from nltk.corpus import stopwords
    print(stopwords.words("english"))

 ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself'
```

Fig 23. Stopwords

```
[ ]  #tokenize the sentence into words
     nltk.download('punkt')
     from nltk.tokenize import word_tokenize

     # Create a list to store tokenized words for each sentence
     tokenized_sentences = []

     # Iterate over each sentence in the 'text' column of the DataFrame 'df'
     for sentence in df['text']:

     # Tokenize the current sentence and store the result in 'words'
         word = word_tokenize(sentence)

      # Append the list of tokenized words to the 'tokenized_sentences' list
         tokenized_sentences.append(word)

     [nltk_data] Downloading package punkt to /root/nltk_data...
     [nltk_data]   Unzipping tokenizers/punkt.zip.
```

Fig 24. Tokenization

```
[ ]  # Lemmatize tokens and convert them to lowercase
     from nltk.stem import WordNetLemmatizer
     from nltk.corpus import wordnet
     nltk.download('wordnet')


     lemmatizer = WordNetLemmatizer()

     [nltk_data] Downloading package wordnet to /root/nltk_data...

[ ]  corpus = []
     for i in range(0, len(tokenized_sentences)):
         review = re.sub('[^a-zA-Z]', ' ', str(tokenized_sentences[i]))
         review = review.lower()
         review = review.split()

         review = [lemmatizer.lemmatize(tokenized_sentences) for tokenized_sentences in review if not tokenized_sentences in stopwords.words('englist
         review = ' '.join(review)
         corpus.append(review)
```

Fig 25. lemmatization

```
[ ] # Preprocess the data using gensim.utils.simple_preprocess
    from gensim.utils import simple_preprocess
    words = [simple_preprocess(sentence) for sentence in corpus]
    words
```

```
  'verbal',
  'communication',
  'creating',
  'delivering',
  'report',
  'presentation',
  'client',
  'deliverable',
  'moreability',
  'quickly',
  'learn',
  'become',
  'skilled',
  'industry',
  'specific',
```

Fig 26. Simple_Preprocess

```
[ ] # Lets train Word2vec
    from gensim.models import Word2Vec
    model = gensim.models.Word2Vec(words,window=5,min_count=2)
```

Fig 27. Train Word2vec Model

```
[ ] # Prepare the Input Data
    X = []
    for sentence in words:
        embedding = []
        for word in sentence:
            try:
                word_embedding = model.wv[word]
                embedding.append(word_embedding)
            except KeyError:
                pass

        # Convert the list of word embeddings for each sentence into a single vector using mean or sum
        sentence_embedding = sum(embedding) / len(embedding) if embedding else []  # Handle empty sentences if necessary
        X.append(sentence_embedding)
```

Fig 28. Word Embedding

```
# Perform padding using pad_sequences
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Define the maximum sequence length for padding.
sent_length = 100

# Perform padding on the sentence embeddings.
padded_X = pad_sequences(X, padding='pre', maxlen=sent_length, dtype='float32')

print(padded_X)
```

```
[[-0.26967415  0.4194918  -0.00512274 ...  0.89018816  0.36787066
  -0.02390319]
 [-0.28566828  0.40047613  0.14509007 ...  0.82003707  0.24309711
  -0.21997043]
 [-0.03878997  0.22186783  0.23509786 ...  0.8422707   0.01500506
   0.37738723]
 ...
 [-0.04745709  0.18929325 -0.09512962 ...  0.73343617  0.05692539
   0.25878263]
 [-0.2060873   0.4893068  -0.25544685 ...  0.20237999  0.41020566
  -0.03620796]
 [-0.53874457  0.5236083   0.08233171 ...  0.65578955  0.38801813
  -0.13755348]]
```

Fig 29. padding

7 Model Building

7.1 Model 1- Long Short-Term Memory (LSTM)

```
## Creating model

# Create the Sequential model
model1=Sequential()

# Add LSTM layer with the desired number of units
model1.add(LSTM(64, input_shape=(1, 100)))

# Add Dense layer
model1.add(Dense(32, activation='relu'))

# Add Dropout layer
model1.add(Dropout(0.3))

#Add output layer
model1.add(Dense(1,activation='sigmoid'))

# Compile the model
model1.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
print(model1.summary())
```

Fig 30. LSTM_Model

```
[ ] # Split the Dataset into Training and Testing Sets
    X_train, X_test, y_train, y_test = train_test_split(X_final, y_final, test_size=0.25, random_state=32)
```

Fig 31. Splitting_Train_Test

```
[ ] # Reshape the input data to have three dimensions
    X_train_reshaped = np.expand_dims(X_train, axis=1)
    X_test_reshaped = np.expand_dims(X_test, axis=1)
```

Fig 32. Reshaped

```
[ ] # Define class weights
    class_weights = {0: 1, 1: 18.64}
```

Fig 33. Class weights

```
[ ] #Training the model
    model1.fit(X_train_reshaped, y_train, batch_size=32, epochs=10, class_weight=class_weights, validation_split=0.2)
```

Fig 34. Model1_Training

```
[ ] # Make predictions
    y_pred = model1.predict(X_test_reshaped)
    y_pred
```

Fig 35. Predict_Test

```
[ ] # Convert probabilities to binary predictions
    threshold = 0.5
    binary_predictions = (y_pred > threshold).astype(int).flatten()
```

Fig 36. Convert_binary

```
[ ] from keras_tuner import HyperModel
    from keras import layers
    from keras import models

    class MyHyperModel(HyperModel):
        def build(self, hp):
            model = models.Sequential()
            model.add(layers.Dense(units=hp.Int('units',
                                                min_value=32,
                                                max_value=512,
                                                step=32),
                                   activation='relu'))
            activation = hp.Choice('activation', values=['relu', 'tanh', 'sigmoid', 'elu'])
            model.add(layers.Activation(activation))
            model.compile(
                optimizer=hp.Choice('optimizer', ['adam', 'sgd', 'rmsprop']),
                loss='sparse_categorical_crossentropy',
                metrics=['accuracy'])
            return model
```

```
[ ] def fit(self, X_train_reshaped, y_train, epochs, validation_split):
            model = self.build()
            history = model.fit(X_train_reshaped, y_train, epochs=epochs, validation_split=validation_split, verbose=0)
            return history.history['val_accuracy'][-1]  # Return the validation accuracy of the last epoch
```

Fig 37. Hyperparameter Tuning (LSTM)

```
[ ] from keras_tuner import Hyperband
    tuner = Hyperband(
        hypermodel=MyHyperModel(),
        loss=['sparse_categorical_crossentropy', 'binary_crossentropy'],
        metrics=['accuracy'],
        directory='my_dir',
        project_name='my_project')
```

```
[ ] # Perform the hyperparameter search for a given number of epochs
    tuner.search(X_train_reshaped, y_train, epochs=10, validation_split=0.2

    # Print the best hyperparameters and the corresponding accuracy
    best_hyperparameters = tuner.get_best_hyperparameters(num_trials=1)[0]
    print("Best Hyperparameters:")
    print(best_hyperparameters.values)
```

Fig 38. Best hyperparameter

```
[ ] # Compile the model
    model1.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])

    # Train the model
    best_model = model1.fit(X_train_reshaped, y_train, epochs=2, validation_split=0.2)
```

Fig 39. Train_model

7.2 Model 2- Bidirectional LSTM (BiLSTM)

```
## Creating model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Bidirectional as biLSTM, LSTM, Dense

# Create the Sequential model
model2=Sequential()

# Add Bidirectional LSTM layer with the desired number of units
model2.add(biLSTM(LSTM(units=128)))

# Add Dense layer
model2.add(Dense(64, activation='relu'))

# Add Dropout layer
model2.add(Dropout(0.3))

#Add output layer
model2.add(Dense(1,activation='sigmoid'))

# Compile the model
model2.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

```
[ ] # Train the BiLSTM
model2.fit(X_train_reshaped, y_train, batch_size=32, epochs=10, class_weight=class_weights, validation_split=0.2)
```

Fig 40. Train_model (BiLSTM)

```
# Make predictions
pred = model2.predict(X_test_reshaped)
pred
```

Fig 41. Predict_test

```
[ ] # Convert probabilities to binary predictions
threshold = 0.5
binary_pred = (pred > threshold).astype(int).flatten()
```

Fig 42. Convert_binary

```
[ ] # Compile the best model
model2.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])

# Train the best model
bst_model = model2.fit(X_train_reshaped, y_train, epochs=2, validation_split=0.2)
```

Fig 43. Train_model

8 Model Evaluation

8.1 Model 1- Evaluation

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Compute the confusion matrix
confusion = confusion_matrix(y_test, binary_predictions)

# Create a ConfusionMatrixDisplay object
disp = ConfusionMatrixDisplay(confusion_matrix=confusion, display_labels=["Genuine", "Fraudulent"])

# Plot the confusion matrix
disp.plot(cmap="Blues")
plt.title("Confusion Matrix")
plt.show()
```

Fig 44. Plot_confusion matrix

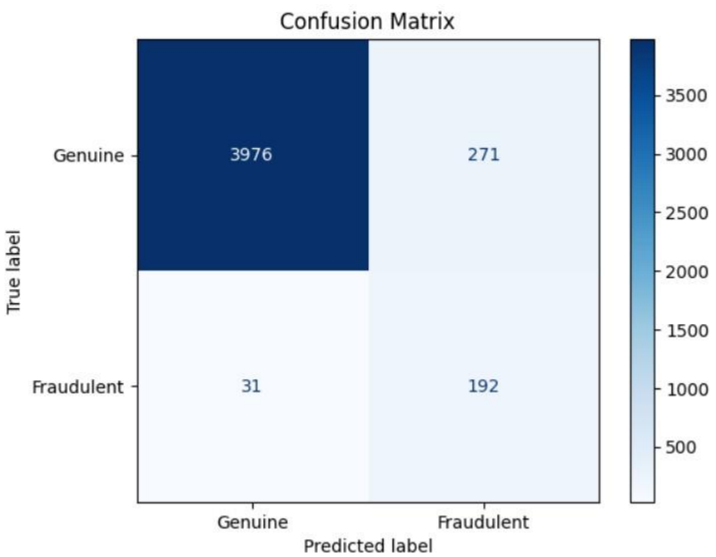

Fig 45. Confusion matrix

```
[ ] # Calculate accuracy
    accuracy_score(y_test,binary_predictions)
    print("Accuracy  {:.3} %".format(accuracy_score(y_test, binary_predictions)*100))

    Accuracy  93.2 %
```

Fig 45. Accuracy_score

```
[ ] from sklearn.metrics import classification_report
    print(classification_report(y_test,binary_predictions))
```

```
              precision    recall  f1-score   support

           0       0.99      0.94      0.96      4247
           1       0.41      0.86      0.56       223

    accuracy                           0.93      4470
   macro avg       0.70      0.90      0.76      4470
weighted avg       0.96      0.93      0.94      4470
```

Fig 46. Classification_report

```
[ ] # Compile the model
    model1.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])

    # Train the model
    best_model = model1.fit(X_train_reshaped, y_train, epochs=2, validation_split=0.2)

    # Evaluate the model on test data
    test_accuracy = model1.evaluate(X_test_reshaped, y_test)
    print("Test Accuracy: ", test_accuracy)
```

```
Epoch 1/2
336/336 [==============================] - 11s 13ms/step - loss: 0.0521 - accuracy:
Epoch 2/2
336/336 [==============================] - 3s 8ms/step - loss: 0.0503 - accuracy: 0
140/140 [==============================] - 1s 5ms/step - loss: 0.0878 - accuracy: 0
Test Accuracy:  [0.08781258016824722, 0.9718120694160461]
```

Fig 47. Evaluate hyperparamter model

```
[ ] # Make predictions on test data
    predictions = model1.predict(X_test_reshaped)

    # Display Predicted Genuine
    print("Predicted Genuine:")
    genuine_count = 0
    for i in range(len(predictions)):
        if predictions[i] < 0.5 and genuine_count < 3:  # Assuming the threshold is 0.5 for binary classification
            print(f"Sample {i+1} - Actual Label: {y_test[i]}, Prediction: {predictions[i][0]}")
            print("Text:", X_test_reshaped[i])
            print("\n")
            genuine_count += 1

    print("\n")

    # Display Predicted Fraudulent
    print("Predicted Fraudulent:")
    fraudulent_count = 0
    for i in range(len(predictions)):
        if predictions[i] >= 0.5 and fraudulent_count < 3:  # Assuming the threshold is 0.5 for binary classification
            print(f"Sample {i+1} - Actual Label: {y_test[i]}, Prediction: {predictions[i][0]}")
            print("Text:", X_test_reshaped[i])
            print("\n")
            fraudulent_count += 1
```

Fig 48. Display_predicted rows

8.2 Model 2- Evaluation

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Compute the confusion matrix
confusion = confusion_matrix(y_test, binary_pred)

# Create a ConfusionMatrixDisplay object
disp = ConfusionMatrixDisplay(confusion_matrix=confusion, display_labels=["Genuine", "Fraudulent"])

# Plot the confusion matrix
disp.plot(cmap="Blues")
plt.title("Confusion Matrix")
plt.show()
```

Fig 49. Plot confusion matrix

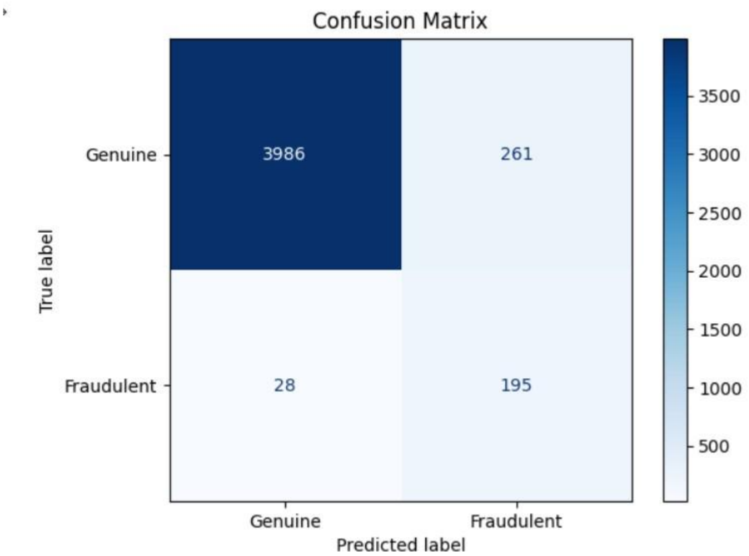

Fig 50. confusion matrix

```
[ ]  # Calculate accuracy
     accuracy_score(y_test,binary_pred)
     print("Accuracy  {:.3} %".format(accuracy_score(y_test, binary_pred)*100))

     Accuracy  93.5 %
```

Fig 52. Accuracy_score

```
[ ] from sklearn.metrics import classification_report
    print(classification_report(y_test,binary_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.94      0.97      4247
           1       0.43      0.87      0.57       223

    accuracy                           0.94      4470
   macro avg       0.71      0.91      0.77      4470
weighted avg       0.96      0.94      0.95      4470
```

Fig 53. Classification_report

```
# Evaluate the model on test data
test_accuracy = model2.evaluate(X_test_reshaped, y_test)
print("Test Accuracy: ", test_accuracy)
```

```
Epoch 1/2
336/336 [==============================] - 11s 17ms/step - los
Epoch 2/2
336/336 [==============================] - 3s 10ms/step - loss
140/140 [==============================] - 1s 6ms/step - loss:
Test Accuracy:  [0.10419961810112, 0.9686800837516785]
```

Fig 54. Evaluate hyperparameter model

```
# Make predictions on test data
predictions = model2.predict(X_test_reshaped)

# Display Predicted Genuine
print("Predicted Genuine:")
genuine_count = 0
for i in range(len(predictions)):
    if predictions[i] < 0.5 and genuine_count < 3:  # Assuming the threshold is 0.5 for binary classification
        print(f"Sample {i+1} - Actual Label: {y_test[i]}, Prediction: {predictions[i][0]}")
        print("Text:", X_test_reshaped[i])
        print("\n")
        genuine_count += 1

print("\n")

# Display Predicted Fraudulent
print("Predicted Fraudulent:")
fraudulent_count = 0
for i in range(len(predictions)):
    if predictions[i] >= 0.5 and fraudulent_count < 3:  # Assuming the threshold is 0.5 for binary classification
        print(f"Sample {i+1} - Actual Label: {y_test[i]}, Prediction: {predictions[i][0]}")
        print("Text:", X_test_reshaped[i])
        print("\n")
        fraudulent_count += 1
```

Fig 55. Display predicted rows

References

Nessa, I. a. (2022). Recruitment Scam Detection Using Gated Recurrent Unit. 2022 IEEE 10th Region 10 Humanitarian Technology Conference (R10-HTC) (pp. 445-449). Hyderabad, India: IEEE.

Srivastava, R. (2022). Identification of Online Recruitment Fraud (ORF). Emirati Journal of Business, Economics and Social Studies, 42 - 54.

Sultana Umme Habiba, M. K. (2021). A Comparative Study on Fake Job Post Prediction Using Different Data mining Techniques. Research Gate.

Tabassum, H. a. (2021). Detecting Online Recruitment Fraud Using Machine Learning. (pp. 472-477). Yogyakarta, Indonesia, 2021: IEEE.