National College of Ireland

Configuration Manual

MSc Research Project
M.Sc. Data Analytics

Jareen Sayma Haque
Student ID: X23383593

School of Computing
National College of Ireland

Supervisor: Abid Yaqoob

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Jareen Sayma Haque
Student ID:	X23383593
Programme:	MSc in Data Analytics **Year:** 2024-2025
Module:	Research Practicum
Lecturer:	Abid Yaqoob
Submission Due Date:	11th August 2025
Project Title:	Using Deep Learning and Transformer Models to Identify Inconsistencies Between Interview Responses and Resume Claims
Word Count:	**Page Count:** 14

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:	Jareen Sayma Haque
Date:	11th August 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

Jareen Sayma Haque
Student ID: X23383593

1. System Configuration

I.1 Hardware

Processor: Intel(R) Core (TM) i7-8550U CPU @ 1.80GHz (1.99 GHz)
Installed RAM 16.0 GB (15.9 GB usable)
System type 64-bit operating system, x64-based processor

I.2 Operating System

Edition Windows 11 Home
Version 24H2
OS build 26100.4652
Experience Windows Feature Experience Pack 1000.26100.128.0

2. Software Requirements

Software's required for build and execution:

- **Integrated Development Environment** : Jupyter Notebook (streamlit_env)
- **Scripting Language** : Python 3.12.3
- **Storage** : Local folder
- **Surveying tool** : Google Form
- **Synthetic data generation tool** : Gemini Flash 3
- **Other Tools** : Notepad ++, Overleaf, Excel

3. Libraries

Various libraries were installed and used in the environment to run the project.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import ast
from sentence_transformers import SentenceTransformer, util
from sentence_transformers.cross_encoder import CrossEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix, roc_curve, precision_recall_curve
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM, AutoModelForSequenceClassification
import torch
from tqdm import tqdm
from sentence_transformers import InputExample
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import seaborn as sns
import lime
from lime.lime_text import LimeTextExplainer
import shap
```

```
import os
os.environ["OMP_NUM_THREADS"] = "1"

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import ast # To parse stringified lists from the CSV
import openpyxl
from wordcloud import WordCloud
from sklearn.feature_extraction.text import CountVectorizer
from scipy.stats import ttest_ind
import textstat

from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.manifold import TSNE
from collections import Counter
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
import nltk
nltk.download('averaged_perceptron_tagger_eng')
sns.set_style('whitegrid')
```

Figure 1 Libraries used

4. Folder Structure

This is the root folder structure below consisting of all the demo artifacts including code, dataset, report and configure manual.

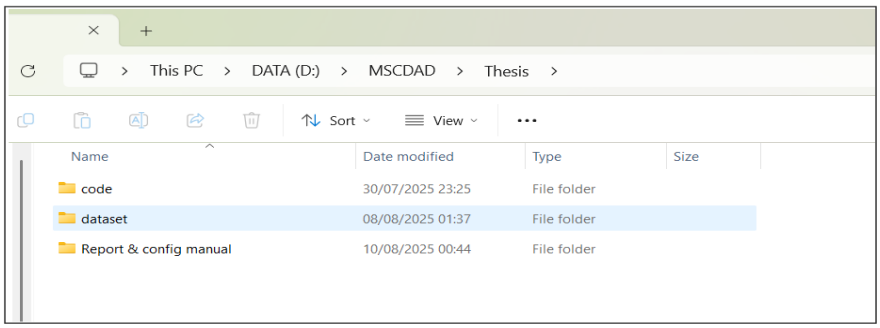


Figure 2 root folder

Inside the code folder, all the codes are structured in 3 subfolders for better clarity,

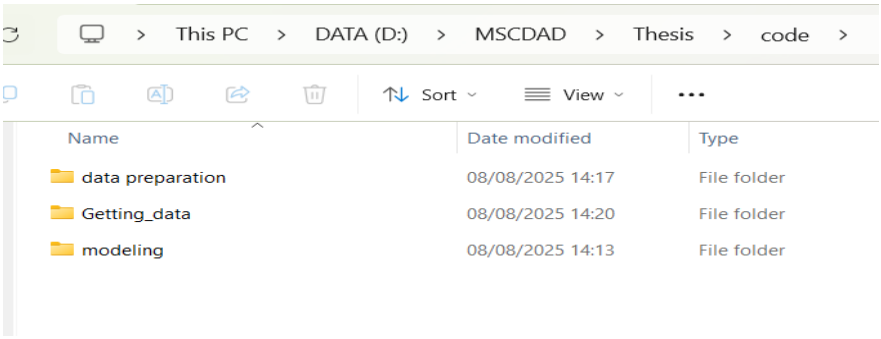


Figure 3 code folder

The dataset folder consists of all the dataset that was used and created throughout the whole procedure.

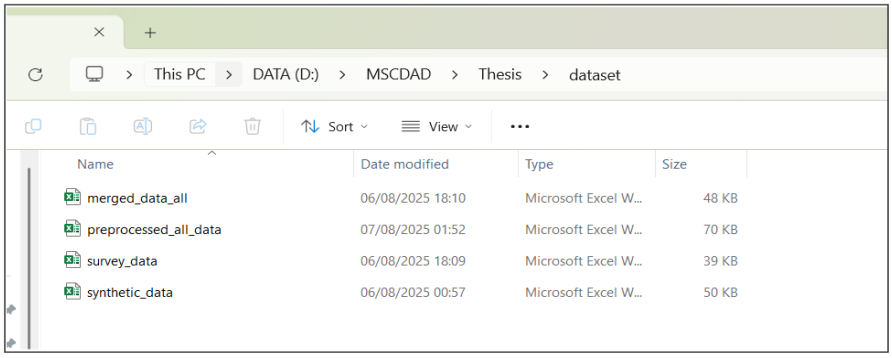


Figure 4 Dataset Folder

Sample of AI_prompt that was used in this data collection stage, is inside the folder below, code ---- getting_data --- “AI_tool_prompt_flash_gemini.docx”

Directory:

The root folder directory structure including all files are given below,

```
root/
├── datasets/
│   ├── survey_data.xlsx
│   ├── merged_data_all.xlsx
│   ├── preprocessed_all_data.xlsx
│   └── synthetic_data.xlsx
├── code/
│   ├── getting_data/
│   │   ├── AI_tool_prompt_flash_gemini.docx
│   │   ├── GeneratingSurvey_Data_Flash.ipynb
│   │   └── mergedfile_alldata.ipynb
│   ├── data_preparation/
│   │   ├── Preprocessing_merged_data_.ipynb
│   │   ├── rawEDA.ipynb
│   │   └── EDA_preprocessed.ipynb
│   └── modeling/
│       └── modeling.ipynb
```

5. Workflow to replicate the experiment

I. Setting Up Your Environment

- Ensure that all the required software and Python libraries are installed.
- Open Jupyter Notebook.
- Set the working directory to the folder containing the dataset files.

II. Data Collection and Augmentation

Primary dataset is collected through google Survey form-
Input file: Survey Form Link,
https://docs.google.com/forms/d/e/1FAIpQLSehZZSICX7eH3elanDJvcUtPqNMpu5XMp33kZ2bdPhg9zSuHA/viewform?usp=sharing&ouid=109136116658387132990

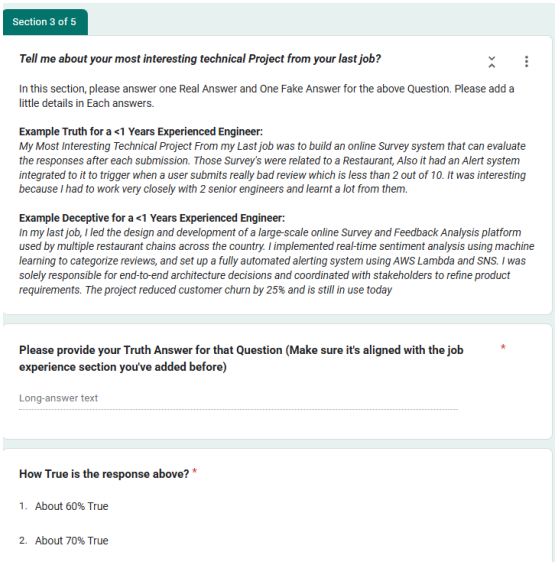


Figure 5 Google Survey Form

from the survey form, dataset is created as an excel file as, Output file: "survey_data.xlsx"

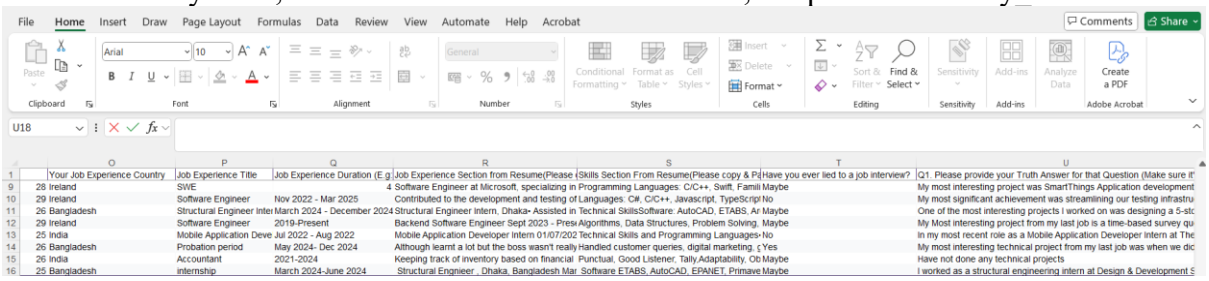
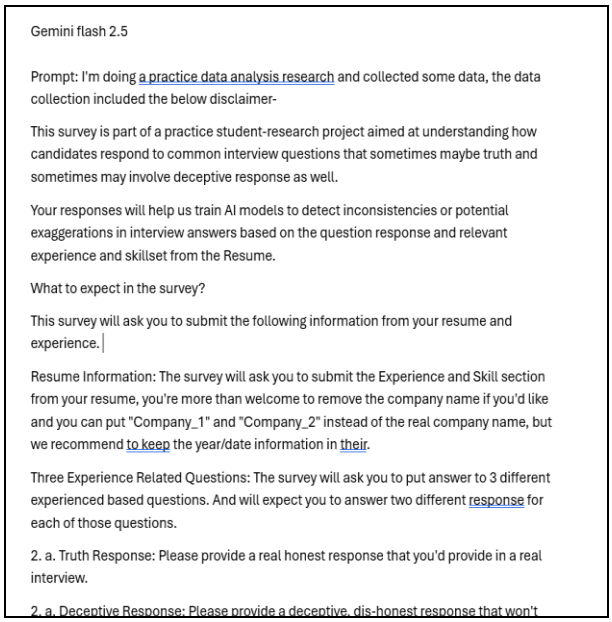


Figure 6 Survey dataset

Gemini flash 2.5 was used as an AI tool to generate synthetic data based on the human survey data excel file, a sample of prompt is below,



Gemini flash 2.5

Prompt: I'm doing a practice data analysis research and collected some data, the data collection included the below disclaimer-

This survey is part of a practice student-research project aimed at understanding how candidates respond to common interview questions that sometimes maybe truth and sometimes may involve deceptive response as well.

Your responses will help us train AI models to detect inconsistencies or potential exaggerations in interview answers based on the question response and relevant experience and skillset from the Resume.

What to expect in the survey?

This survey will ask you to submit the following information from your resume and experience.

Resume Information: The survey will ask you to submit the Experience and Skill section from your resume, you're more than welcome to remove the company name if you'd like and you can put "Company_1" and "Company_2" instead of the real company name, but we recommend to keep the year/date information in their.

Three Experience Related Questions: The survey will ask you to put answer to 3 different experienced based questions. And will expect you to answer two different response for each of those questions.

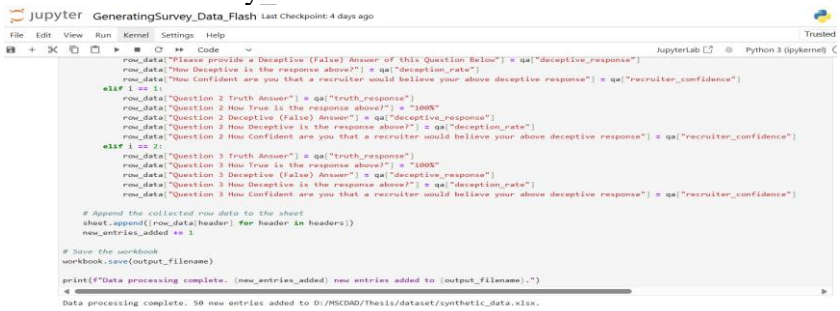
2. a. Truth Response: Please provide a real honest response that you'd provide in a real interview.

2. a. Deceptive Response: Please provide a deceptive, dis-honest response that won't

Figure 7 AI prompt sample

Based on the survey dataset, Synthetic dataset is generated through Few-shot learning-based Data synthesis using Gemini flash 2.5 prompt and python into an excel file. Files used are listed below,

- Input file: Prompt sample: " \AI_tool_prompt_flash_gemini.docx"
- Process file: "\code\Getting_data\GeneratingSurvey_Data_Flash.ipynb"
- Output file: "dataset\survey_data.xlsx"



jupyter GeneratingSurvey_Data_Flash Last Checkpoint: 4 days ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

```
            row_data["Please provide a Deceptive (False) Answer of this Question Below"] = qa["deceptive_response"]
            row_data["How Deceptive is the response above?"] = qa["deception_rate"]
            row_data["How Confident are you that a recruiter would believe your above deceptive response"] = qa["recruiter_confidence"]
        elif i == 1:
            row_data["Question 2 Truth Answer"] = qa["truth_response"]
            row_data["Question 2 How True is the response above?"] = "100%"
            row_data["Question 2 Deceptive (False) Answer"] = qa["deceptive_response"]
            row_data["Question 2 How Deceptive is the response above?"] = qa["deception_rate"]
            row_data["Question 2 How Confident are you that a recruiter would believe your above deceptive response"] = qa["recruiter_confidence"]
        elif i == 2:
            row_data["Question 3 Truth Answer"] = qa["truth_response"]
            row_data["Question 3 How True is the response above?"] = "100%"
            row_data["Question 3 Deceptive (False) Answer"] = qa["deceptive_response"]
            row_data["Question 3 How Deceptive is the response above?"] = qa["deception_rate"]
            row_data["Question 3 How Confident are you that a recruiter would believe your above deceptive response"] = qa["recruiter_confidence"]

    # Append the collected row data to the sheet
    sheet.append([row_data[header] for header in headers])
    new_entries_added += 1

# Save the workbook
workbook.save(output_filename)

print(f"Data processing complete. {new_entries_added} new entries added to {output_filename}.")
```

```
Data processing complete. 50 new entries added to D:/MSCDAD/Thesis/dataset/synthetic_data.xlsx.
```

Figure 8 Generating synthetic data snippet

Synthetic data generated,

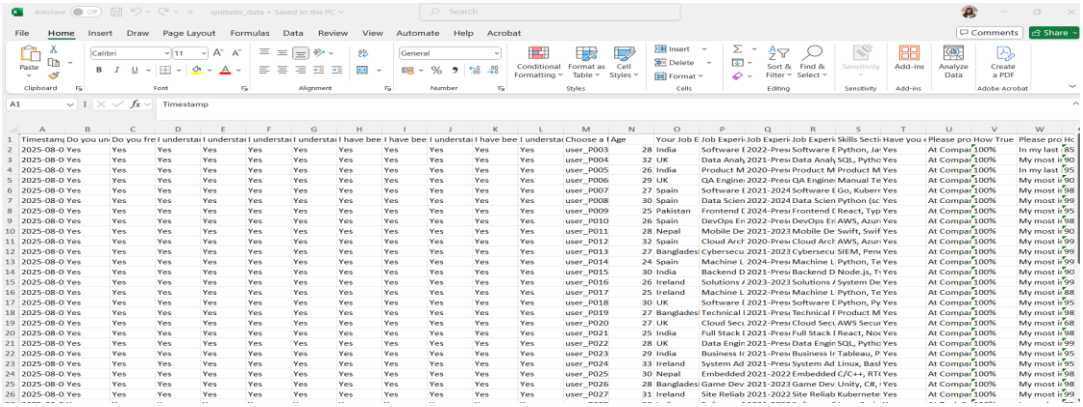


Figure 9 Synthetic Dataset

Data is merged programmatically into excel file. Files used in this stage are below,

- Input file: " \code\Getting_data\mergedfile_alldata.ipynb"
- Output file: " \dataset\merged_data_all.xlsx"

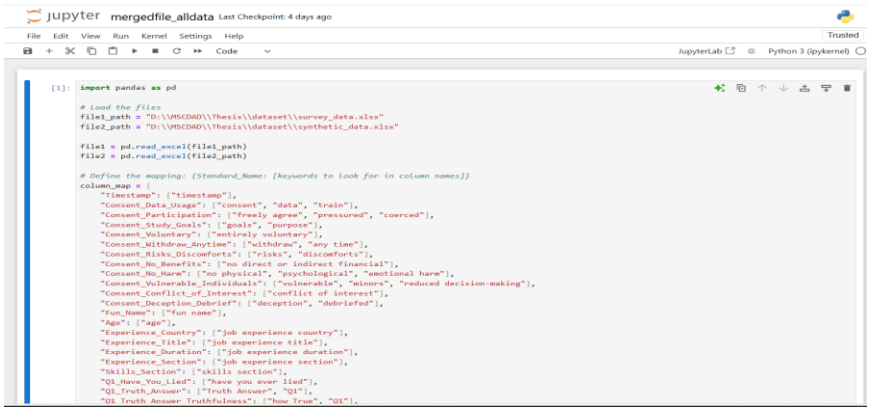


Figure 10 Merging dataset code snippet

Merged dataset,

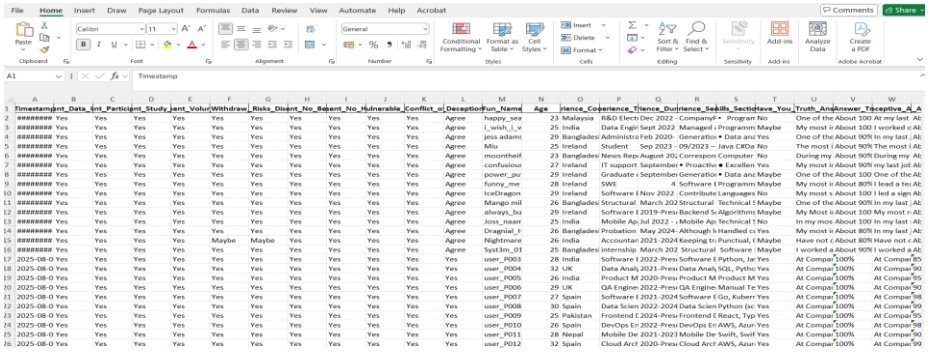


Figure 11 Merged Dataset

III. Pre-Processing & EDA

Merged dataset was pre-processed to make it ready for model training. Also 2 step EDA was done on the dataset, before and after preprocessing, to derive better and clearer insights from the data.

Data is pre-processed and stored in a excel file. Files used are below,

- Input file: "code\data preparation\Preprocessing_merged_data_.ipynb”

```python
def preprocess_text_pipeline(text_series, include_stopwords=False):
    """
    Applies the full preprocessing pipeline to a pandas Series of text.

    Args:
        text_series (pd.Series): The pandas Series containing text data.
        include_stopwords (bool): If True, stop words will NOT be removed.
                                  Useful for tasks where context is critical.

    Returns:
        pd.Series: A Series of preprocessed text (list of lemmatized tokens).
    """
    print(f"Starting preprocessing for series: {text_series.name if hasattr(text_series, 'name') else 'Unnamed Series'}")
     # Step 1: Handle Missing Values
    print("  - Handling missing values...")
    processed_series = handle_missing_values(text_series)

    # Step 2: Normalize Text (lowercase, remove punctuation/special chars)
    print("  - Normalizing text (lowercasing, removing special characters)...")
    processed_series = processed_series.apply(normalize_text)

    # Step 3: Tokenization
    print("  - Tokenizing text...")
    processed_series = processed_series.apply(tokenize_text)

    # Step 4: Stop Word Removal (conditional)
    if not include_stopwords:
        print("  - Removing stop words...")
        processed_series = processed_series.apply(remove_stopwords)
    else:
        print("  - Skipping stop word removal (as requested)...")

    # Step 5: Lemmatization
    print("  - Lemmatizing tokens...")
    processed_series = processed_series.apply(lemmatize_tokens)

    print(f"Finished preprocessing for series: {text_series.name if hasattr(text_series, 'name') else 'Unnamed Series'}\n")
    return processed_series
```

Figure 12 Data Pre-processing

Output file: "dataset\preprocessed_all_data.xlsx"

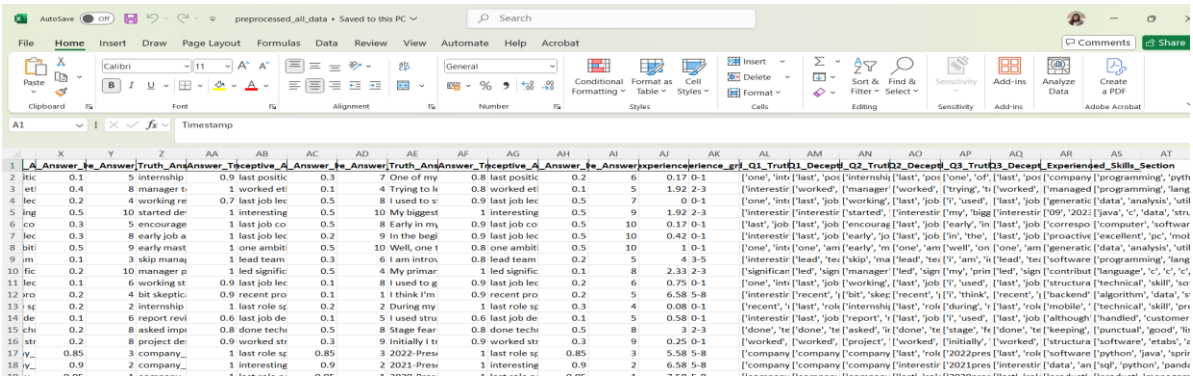


Figure 13 Preprocessed data excel file

EDA:

Exploratory data analysis is performed through these two code files,

EDA on dataset **Before** Pre-processed:

- Input file:
  - I. "dataset \interview_data-.xlsx"
  - II. “dataset\Interview Response Survey (Responses).xlsx"
- Process: "code\data preparation\rawEDA.ipynb"

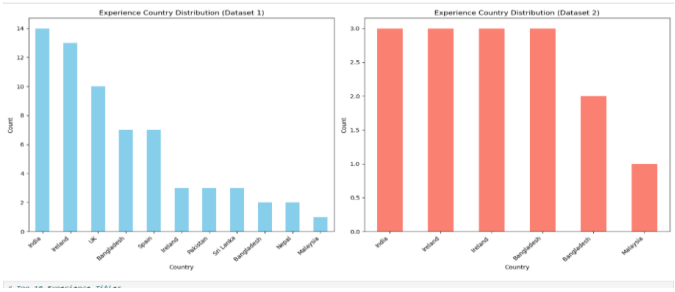


Figure 14 sample EDA snippet on raw datasets

EDA on dataset **After** Pre-processed:

- Input file: " dataset\preprocessed_all_data.xlsx"
- Process: "code\data preparation\EDA_preprocessed.ipynb"

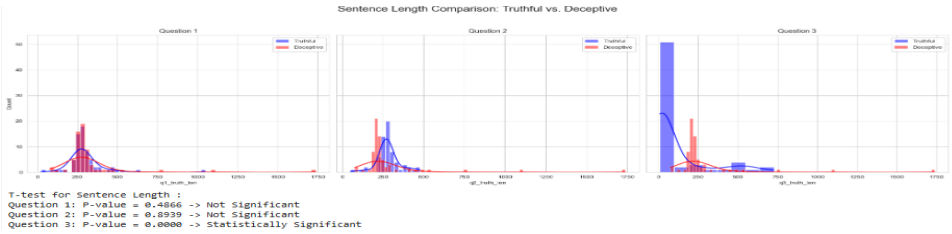


Figure 15 Sample EDA on pre-processed data

IV. Feature Engineering and Data Splitting

Pre-processed data set is loaded to feature engineering pipeline,

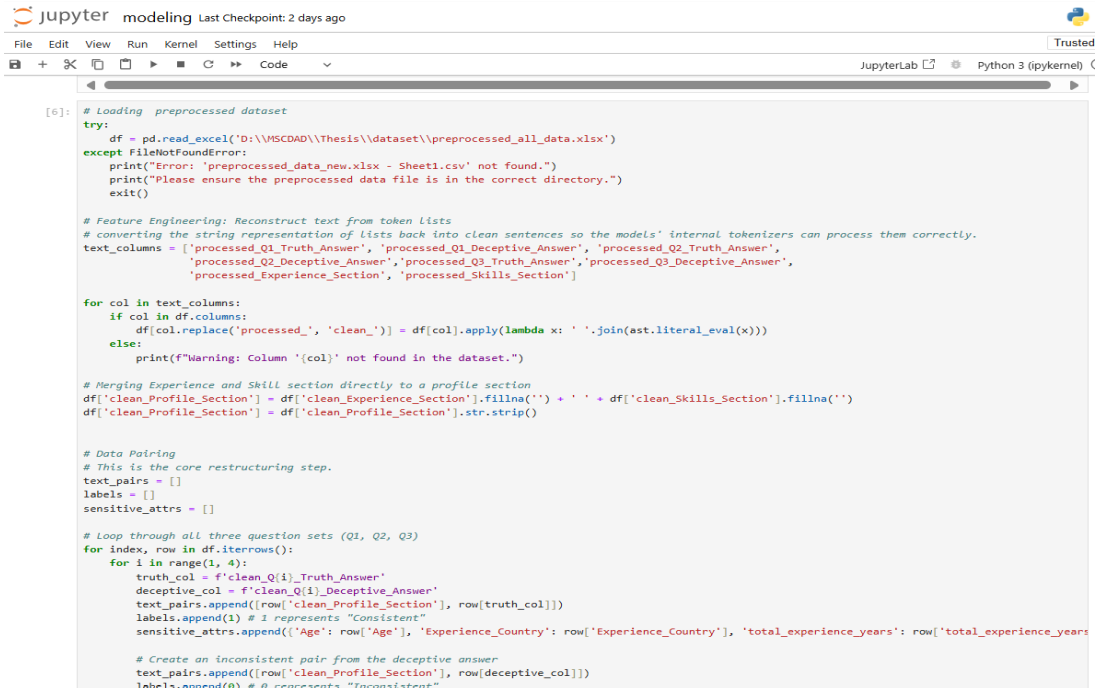


Figure 16 Feature engineering snippet

Then dataset is splitted through stratify to 80-20 ratio to prepare it for model training stage,

```
# data split into training and testing sets
X_train, X_test, y_train, y_test, sensitive_train, sensitive_test = train_test_split(
    full_df[['resume_segment', 'interview_answer']],
    full_df['label'],
    full_df[['Age', 'Experience_Country', 'total_experience_years']],
    test_size=0.2,
    random_state=42,
    stratify=full_df['label']
)

print(f"Total pairs created for modeling: {len(full_df)}")
print(f"Training set size: {len(X_train)}")
print(f"Test set size: {len(X_test)}")
print("\nSample Data Point:")
print(X_train.iloc[0])
print(f"Label: {y_train.iloc[1]}")
print("\nAnother Sample Data Point:")
print(X_train.iloc[1])
print(f"Label: {y_train.iloc[1]}")

y_true = np.asarray(y_test).ravel().astype(int)
print("Test positives:", y_true.sum(), "negatives:", (1-y_true).sum())
```

```
Total pairs created for modeling: 390
Training set size: 312
Test set size: 78
```

Figure 17 Dataset splitting

V. Model Training

The splitted data went through 3 architecture frameworks through rigorous model fine tuning,

Architectural Frameworks	Models Tested	Libraries
Bi-Encoder (SBERT)	all-MiniLM-L-6-v2 stsb-roberta-base	Hugging Face – *Sentence-Transformers*
Cross-Encoder (MiniLM)	cross-encoder/ms-marco-MiniLM-L-6-v2	Hugging Face – *Transformers* + *Sentence-Transformers*
Natural Language Inference (T5)	google/flan-t5-small	Hugging Face – *Transformers*

Final model, The pre-trained cross-encoder/ms-marco-MiniLM-L-6-v2, was loaded and fine-tuned on the training data. The process was configured to run for 4 epochs with a batch size of 4, warm-up period of 10 steps was used to stabilize the learning rate, and the maximum input sequence length was 384 tokens.

```
# Cross-Encoder using ms-marco-MiniLM-L-6-v2 model
# Setting max_length=384 to reduce the load to PC, our texts are aligned wiht this max token length

# Prepare training examples
train_samples = [InputExample(texts=[row.resume_segment, row.interview_answer], label=float(row.label))
                 for row in X_train.join(y_train).itertuples(index=False)]

cross_encoder_model = CrossEncoder("cross-encoder/ms-marco-MiniLM-L-6-v2", num_labels=1, max_length=384)

# DataLoader
train_dataloader = DataLoader(train_samples, shuffle=False, batch_size=4)

# Train
cross_encoder_model.fit(train_dataloader=train_dataloader,
          epochs=4,
          warmup_steps=10)


preds = cross_encoder_model.predict(list(zip(X_test['resume_segment'], X_test['interview_answer'])))

# Convert to class labels (threshold)
pred_labels = (np.array(preds) >= 0.5).astype(int)

#Evaluate
print("🔍 Cross-Encoder Evaluation Results")
print("Accuracy:", accuracy_score(y_test, pred_labels))
print("Precision:", precision_score(y_test, pred_labels))
print("Recall:", recall_score(y_test, pred_labels))
print("F1 Score:", f1_score(y_test, pred_labels))
print("ROC-AUC:", roc_auc_score(y_test, preds))
print("Confusion Matrix:\n", confusion_matrix(y_test, pred_labels))
```

Figure 18 Cross encoder training snippet

Other models like SBert & t5 went through threshold tuning to test accuracy,

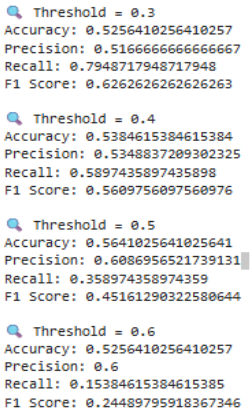


Figure 19 SBert Threshold Tuning

VI. Evaluation

Each model evaluations metrics were tested where cross encoder performed best.

Cross-Encoder	**Bi-Encoder**	**T5 NLI**
Accuracy: 0.7307692307692307 Precision: 0.7368421052631579 Recall: 0.717948717948718 F1 Score: 0.7272727272727273 ROC-AUC: 0.836620644312952 Confusion Matrix: [[29 10] [11 28]]	1. All-MiniLM-L-6-v2 Accuracy: 0.5 Precision: 0.0 Recall: 0.0 F1 Score: 0.0 ROC-AUC: 0.5 Confusion Matrix: [[39 0] [39 0]] 2. Stsb-roberta-base Accuracy: 0.5256410256410257 Precision: 0.625 Recall: 0.1282051282051282 F1 Score: 0.2127659574468085 ROC-AUC: 0.5256410256410257 Confusion Matrix: [[36 3] [34 5]]	Accuracy: 0.5897435897435898 Precision: 1.0 Recall: 0.1794871794871795 F1 Score: 0.30434782608695654 ROC-AUC: 0.5897435897435898 Confusion Matrix: [[39 0] [32 7]]

Cross-encoders model evaluation is also showed through dashboard using matplotlib and seaborn,

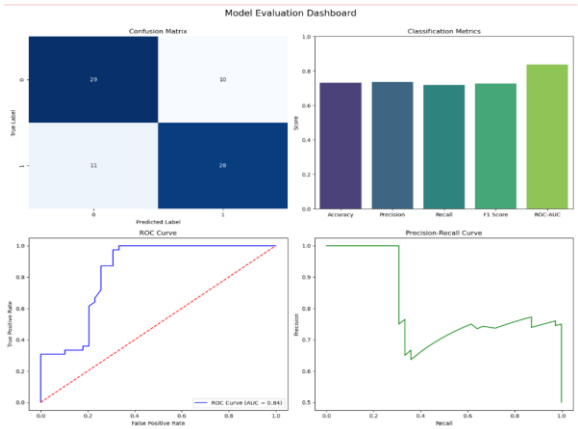


Figure 20 Cross-Encoder Model Evaluation Dashboard

VII. Fairness & Bias

Fairness audits were also performed, and SHAP/Lime Libraries were installed and used to show bias detection and mitigation through 2 examples from the result.

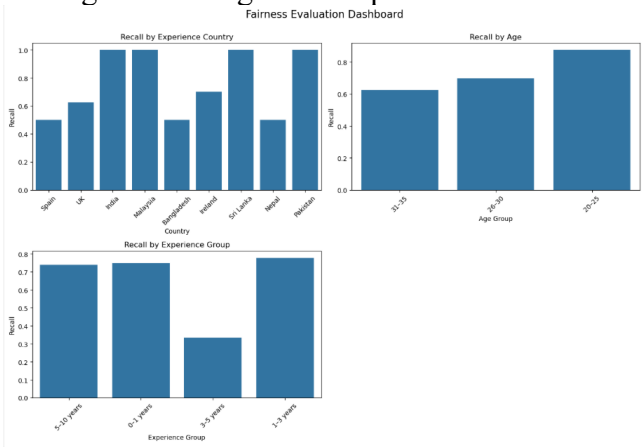


Figure 21 Fairness dashboard

```python
lime_model_name = "cross-encoder/ms-marco-MiniLM-L-6-v2"
tokenizer = AutoTokenizer.from_pretrained(lime_model_name)
lime_model = AutoModelForSequenceClassification.from_pretrained(lime_model_name)


def predict_proba(text_pairs, batch_size=16):
    all_probs = []
    for i in range(0, len(text_pairs), batch_size):
        batch_pairs = text_pairs[i:i+batch_size]
        encodings = tokenizer.batch_encode_plus(
            batch_pairs, padding=True, truncation=True, return_tensors="pt"
        )
        with torch.no_grad():
            outputs = lime_model(**encodings)
            logits = outputs.logits  # Shape: (batch_size, 1)
            scores = torch.sigmoid(logits).cpu().numpy()  # Convert to [0, 1] range

            # Convert to 2-class probability: [No Match prob, Match prob]
            probs = np.hstack([1 - scores, scores])  # Shape: (batch_size, 2)
            all_probs.extend(probs)
    return np.array(all_probs)
```

```python
import lime
import numpy as np
from lime.lime_text import LimeTextExplainer
class_names = ["No Match", "Match"]
explainer = LimeTextExplainer(class_names=class_names)
sample_data = full_df[['resume_segment', 'interview_answer']].head(2)
print(sample_data)

for idx, row in sample_data.iterrows():
    resume_text = row['resume_segment']
    interview_text = row['interview_answer']
```

Figure 22 Lime snippet

```python
import shap
#  Pre-computation Step
shap.initjs()
sample_data_short = full_df[['resume_segment', 'interview_answer']].head(2)
plt.figure(figsize=(20, 20))  # Width=20 inches, Height=20 inches (adjust as needed)

#  Loop through your data
# Using 'break' at the end to demonstrate with just one example
for idx, row in sample_data_short.iterrows():
    resume_text = row['resume_segment']
    interview_text = row['interview_answer']

    # 1. GET PREDICTION DATA FIRST
    # Run a prediction on the text to get the probabilities and predicted class
    probabilities = predict_proba([(resume_text, interview_text)])[0]
    predicted_class_index = np.argmax(probabilities)

    predicted_class_name = class_names[predicted_class_index]
    confidence_score = probabilities[predicted_class_index]

    # 2. PRINT PREDICTION CONTEXT
    print("--- Prediction Details ---")
    print('--- Original Resume Experience ---')
    print(df['Experience_Section'][0])
    print('--- Original Resume Skills ---')
    print(df['Skills_Section'][0])

    print(f"--- CLEANED TEXT ANALYZED: --- '{interview_text}'")
    print(f"--- PREDICTED CLASS: ---'{predicted_class_name}' (Class Index: {predicted_class_index})")
    print(f"--- CONFIDENCE SCORE: ---{confidence_score:.4f}")
    print("-" * 50 + "\n")


    # 3. RUN SHAP EXPLAINER
    def shap_predict(texts):
        text_pairs = [(resume_text, text) for text in texts]
        return predict_proba(text_pairs)

    explainer = shap.Explainer(shap_predict, tokenizer)
    shap_values = explainer([interview_text])
```

Figure 23 SHAP code snippet

Through the Lime and SHAP framework, Bias is explained,

Figure 24 LIME Explainability Generated

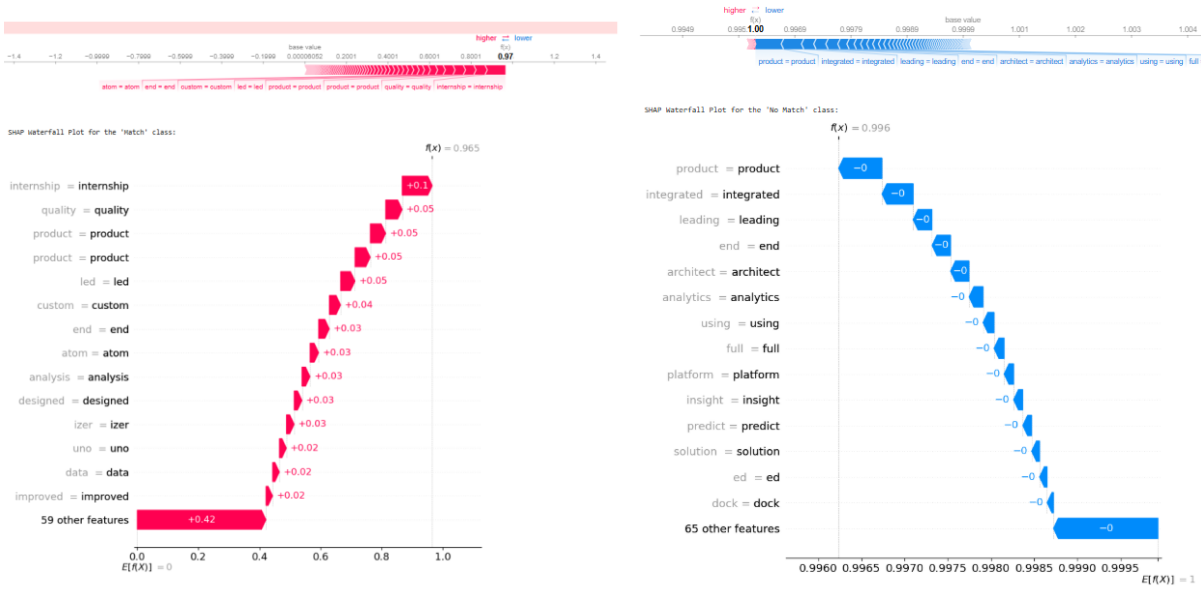


Figure 25 SHAP Explanation Generated

6. Possible Conflicts & Troubleshooting:

I. CPU Limitation:

To align with possible CPU limitations, might need to limit the CPU usage to not use multithreading, for which configured the code to use single thread using the following:

```
Import os
os.environ["OMP_NUM_THREADS"] = "1"
os.environ["TOKENIZERS_PARALLELISM"] = "false"
os.environ["MKL_NUM_THREADS"] = "1"
os.environ["KMP_INIT_AT_FORK"] = "FALSE"
```

II. Library Conflict:

Conflicting version of some libraries could cause error in the environment 'streamlit_env', for which the specific version of these libraries are recommended to replicate the project to ensure smooth execution. A few lists are given below

Libraries	Version
pip	24.0
nltk	3.8.1
pandas	2.2.2
numpy	1.26.4

III. Notebook Crashing - Environment Preflight Test

To prevent system crashes, this preflight test code below should be run **before** training on pre-processed data inside the IDE in testing device. This will help to determine the optimal **model**, **device type**, and encoder 'max_length' that works with PC's hardware configuration and limitations.

```
bi_encoder_name = "sentence-transformers/all-MiniLM-L6-v2"   # Bi-Encoder model
bi = SentenceTransformer(bi_encoder_name, device="cpu") # Chose device="cuda" if needed
bi.max_seq_length = 256 # reduce memory

# Use a 1-output head model:
ce_model_name = "cross-encoder/ms-marco-MiniLM-L-6-v2"   # small & fast; single score
ce = CrossEncoder(ce_model_name, num_labels=1, device="cpu", max_length=256)

sample_resume = str(X_train.iloc[0]["resume_segment"])
sample_answer = str(X_train.iloc[0]["interview_answer"])

_ = bi.encode([sample_resume, sample_answer], show_progress_bar=False)
_ = ce.predict([(sample_resume, sample_answer)], convert_to_numpy=True)

print("Preflight OK.")
```

This configuration manual provides the necessary steps and instructions for setting up, executing, and understanding the implementation of the Interview inconsistency detection framework.

References

Google Survey link:
https://docs.google.com/forms/d/e/1FAIpQLSehZZSICX7eH3elanDJvcUtPqNMpu5XMp33kZ2bdPhg9zSuHA/viewform?usp=sharing&ouid=109136116658387132990