National College of Ireland

Configuration Manual

MSc Research Project

NLP and Machine Learning Framework for Reducing False Positives in Watchlist Name Screening in AML Systems

Gayatri Jadhav
Student ID: 23407476

School of Computing
National College of Ireland

Supervisor: Hamilton Niculescu

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Gayatri Jadhav

Student ID: 23407476

Year Programme:
2025-2026

Module: **MSc (Research) Practicum**

Lecturer: Hamilton Niculescu

Submission Due
Date: 28/01/2026

Project Title: MSc Research Project
NLP and Machine Learning Framework for Reducing False Positives in Watchlist Name Screening in AML Systems

Word Count: 1492 **Page Count:** 20

I hereby certify that the information contained in this (my submission) is information about 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 another author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Gayatri Jadhav

Date: 23/01/2026

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 a computer.	□

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

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

Configuration Manual

Gayatri Jadhav

Student ID: 23407476

1 System Requirement and Setup

All experiments in this dissertation were executed locally using Jupyter Notebook, managed within an Anaconda (conda) environment. This setup provided a controlled, reproducible environment suitable for natural language processing, machine-learning experimentation, and model explainability using SHAP.
The following hardware and software configurations were used to ensure full reproducibility of the results.

1.1 Hardware Configuration

The project was developed and executed on a standard personal workstation using virtualised environments provided by Anaconda.
Hardware Specifications Used:

- **CPU:** Intel® Core™ i5 Processor (Quad-Core)
- **RAM:** 16 GB
- **Disk Storage:** 256 GB SSD (with ~20 GB available for datasets and model caching)
- **Graphics:** No GPU required (all models were executed on CPU)
- **Environment:** Local Jupyter Notebook (browser-based interface)

This configuration is sufficient to support:

- SBERT embedding generation (inference only)
- Fuzzy string-similarity computations
- Training and evaluation of lightweight ML classifiers (Random Forest, XGBoost, SVM)
- SHAP explainability visualisations
- Data preprocessing and exploratory analysis

Because the project uses **only three engineered similarity features** and does not perform any transformer model fine-tuning, CPU-only execution is efficient and fully adequate.

1.2 Software Configuration

All development and experimentation were carried out using the following software stack:

```
import pandas as pd
import xml.etree.ElementTree as ET
import random
from faker import Faker
import unidecode
from fuzzywuzzy import fuzz
from fuzzywuzzy import process
from sentence_transformers import SentenceTransformer, util
import torch
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC
import shap
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_curve, auc, confusion_matrix, classification_report, ConfusionMatrixDisplay, roc_auc_score
```

Figure 1: Importing Libraries

Platform & Environment

- Development Platform: Jupyter Notebook
- Environment Manager: Anaconda (Conda environment)
- Programming Language: Python 3.10
- Operating System: Windows 11 (local machine)
- Compatible with macOS and Linux (Ubuntu 20.04+)

Core Libraries Used

Data Handling & Processing

- pandas
- NumPy
- xml.etree.ElementTree (for UN sanctions list parsing)
- Faker – synthetic customer profile generation
- unidecode and random – Unicode normalisation and sampling

Similarity & NLP Components

- Rapidfuzz (TokenSetRatio, PartialRatio string similarity)
- sentence-transformers (Sentence-BERT embeddings)

Machine Learning Models

- RandomForestClassifier
- SVM (Support Vector Machine) – RBF Kernel
- GridSearchCV – hyperparameter tuning
- XGBClassifier – gradient-boosted tree model

Model Explainability

SHAP - (SHAP KernelExplainer and visualisations)

Visualisation

- matplotlib
- seaborn
-

1.3 Creating the Project Environment

A special Conda environment is developed to make the project reproducible and prevent dependency conflicts. It is an environment that contains all the necessary libraries, such as pandas, NumPy, Rapidfuzz, sentence-transformers, scikit-learn, XGBoost and SHAP.

Step 1: Create a new Conda environment

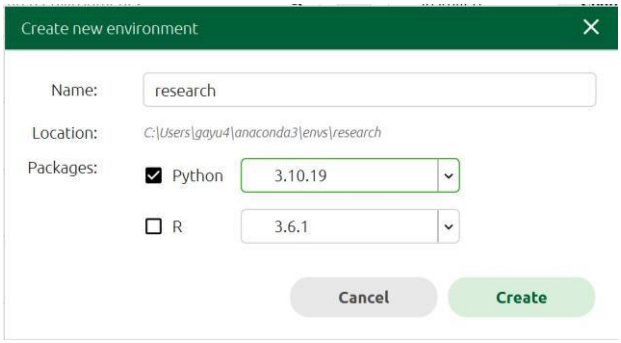


Figure 1: Conda Environment Creation

Step 2: Activate the environment

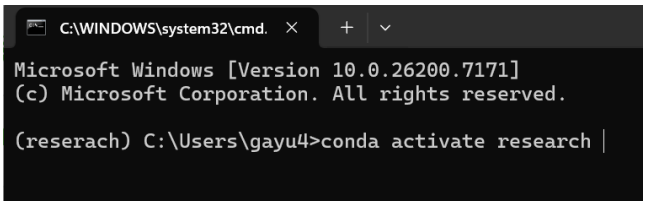


Figure 2: Activating the environment

Step 3: Install essential libraries

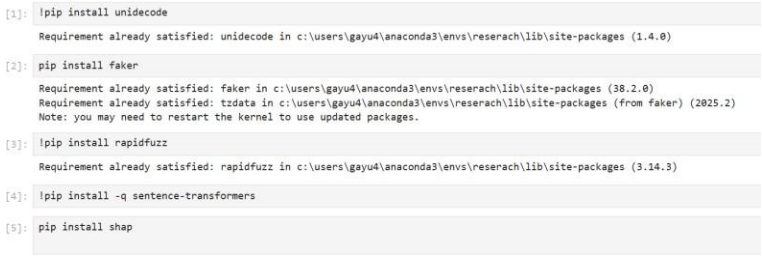


Figure 3: Installing Libraries

Step 4: Downloading the Datasets

Download the datasets from the link mentioned in section 2.1 and save them in a single folder. This Folder path should be passed in juypter notebook while loading the datasets as mention in Figure 4, 5 and 6.

```
# ------------------------------
# 1. Load EU Sanctions (CSV) with correct delimiter
# ------------------------------
try:
    eu_df = pd.read_csv(r"C:\Users\gayu4\Downloads\20251121-FULL-1_1.csv", delimiter=";")
except Exception:
    eu_df = pd.read_csv(r"C:\Users\gayu4\Downloads\20251121-FULL-1_1.csv", delimiter=",")  # fallback

print("EU dataset shape:", eu_df.shape)
print(eu_df.head())
```

Figure 4: Loading EU dataset

```
# ------------------------------
# 2. Load OFAC SDN List (CSV)
# ------------------------------
ofac_df = pd.read_csv(r"C:\Users\gayu4\Downloads\sdn.csv")
print("OFAC dataset shape:", ofac_df.shape)
print(ofac_df.head())
```

Figure 5: Loading OFAC Dataset

```
# ------------------------------
# 3. Load UN Consolidated List (XML)
# ------------------------------
tree = ET.parse(r"C:\Users\gayu4\Downloads\consolidatedLegacyByPRN.xml")
root = tree.getroot()
```

Figure 6: Loading UN Consolidated List

2 Dataset Description

This project utilised two primary data sources: (1) publicly available international sanctions lists, and (2) synthetic customer-name data generated using using the Faker library. No real customer information was used at any stage, ensuring full compliance with privacy and ethical standards.

The combined dataset was used to create labelled name pairs (match / non-match) for training and evaluating the hybrid NLP–ML classification framework.

2.1 Source

Three major global sanctions lists were used as the authoritative dataset from which sanctioned entity names and aliases were extracted:

1. OFAC – Specially Designated Nationals (SDN) List

- Format: CSV
- Content: sanctioned individuals, organisations, known aliases, and transliteration variants.
- Link: https://www.treasury.gov/ofac/downloads/sdn.csv

2. United Nations Consolidated Sanctions List

- Format: XML
- Content: individuals and entities with multiple name components (first/middle/family names) and structured aliases.
- Link: https://scsanctions.un.org/resources/xml/en/consolidated.xml

3. European Union (EU) Sanctions List

- Format: CSV
- Content: consolidated list of sanctioned persons, groups, and entities, including alternative spellings.

- Link: https://webgate.ec.europa.eu/fsd/fsf/public/files/csvFullSanctionsList_1_1/content?token=dG9rZW4tMjAxNw

Watchlist Cleaning & Integration

Using Python preprocessing scripts, the following steps were performed:

- Parsing XML and CSV structures
- Extracting primary names and alias names
- Normalising unicode characters for multilingual consistency
- Removing duplicates, blanks, and non-name artefacts
- Combining all three sources into a single master sanctions database

This consolidated watchlist served as the **reference list** for generating meaningful name-pair combinations.

2.2 Synthetic Customer Dataset

A synthetic customer dataset was generated programmatically to avoid the use of real customer information and ensure full compliance with privacy and ethical requirements. The Faker Python library was used to create artificial customer profiles that resemble realistic naming and demographic patterns.

Data Fields Generated

Each synthetic customer record included:

- **CustomerName**
- **Country**
- **DateOfBirth**
- **Address**

These fields were generated automatically using Faker, ensuring diversity and realism without exposing any personal data.

Injection of Sanctioned Names (2%)

To create a small number of *true match* cases:

- 2% of customer names were randomly replaced with names sampled from the combined sanctions dataset.
- The remaining 98% were fully synthetic Faker-generated names.

This helped maintain a controlled balance of match and non-match labels for later model training

```python
import pandas as pd
import random
from faker import Faker

fake = Faker()

# Number of synthetic customers
n = 1000

data = []
for _ in range(n):
    # 2% chance to pick a sanctioned name for testing
    if random.random() < 0.02:
        name = master['Name'].sample(1).values[0]  # use 'master' instead of 'master_info'
        country = master[master['Name'] == name]['Country'].values[0]
    else:
        name = fake.name()
        country = fake.country()

    data.append({
        "CustomerName": name,
        "Country": country,
        "DateOfBirth": fake.date_of_birth(minimum_age=18, maximum_age=80),
        "Address": fake.address()
    })

synthetic_df = pd.DataFrame(data)
synthetic_df.head()
```

	CustomerName	Country	DateOfBirth	Address
0	Randall Mcdonald	Jamaica	1999-08-17	02121 Turner Cliff Suite 508\nKingtown, FM 68251
1	Bill Stephens	China	1957-09-01	3211 Melissa Shoal\nLyonsmouth, VI 80089
2	Rachel Jackson	San Marino	1962-09-27	104 Travis Greens Suite 989\nBoydfurt, ID 37742
3	David Nelson	Saint Lucia	1977-06-02	Unit 4294 Box 5720\nDPO AP 54906
4	Amy Delacruz	Lesotho	1947-04-30	6002 Chad Burgs Apt. 656\nLake Amanda, DC 62039

Figure 7: Synthetic data

2.3 Formatting, Cleaning and Standardisation

All preprocessing steps in this project were executed programmatically inside the Jupyter Notebook to ensure consistency, reproducibility, and the absence of manual intervention. Both the sanctions dataset and the synthetic customer dataset were passed through a uniform cleaning workflow.

Automated Cleaning Procedures

- Basic Normalisation

  Lowercasing and Unicode stripping were applied to create consistent clean-name fields across datasets.

2.4 Combined Name-Pair Dataset Construction

After the sanctions dataset and synthetic customer dataset were cleaned and standardised, a combined name-pair dataset was created. This dataset forms the input for all subsequent similarity computations and machine-learning models.

Pair Generation Process

The following steps were performed programmatically in Jupyter Notebook:

- Matched pairs (Label = 1) were created by pairing synthetic customer names with randomly sampled sanctioned names
- Non-matched pairs (Label = 0) were created by pairing the same customer names with different randomly selected sanctioned names.

[27]:
```
# Assume 'synthetic_df' and 'master' have 'CleanName' columns
positive_pairs = []
negative_pairs = []

for _, row in synthetic_df.iterrows():
    # Positive pair – same or close name
    sanctioned_match = master.sample(1).iloc[0]
    positive_pairs.append((row['CleanName'], sanctioned_match['CleanName'], 1))

    # Negative pair – random mismatch
    sanctioned_nonmatch = master.sample(1).iloc[0]
    negative_pairs.append((row['CleanName'], sanctioned_nonmatch['CleanName'], 0))

pairs_df = pd.DataFrame(positive_pairs + negative_pairs, columns=['CustomerName', 'SanctionName', 'Label'])
```

Figure 8: Programmatic Construction of Labeled Customer–Sanctions Name Pairs

3 Feature Engineering Setup

This section describes how similarity features were generated for each customer–sanction name pair after the combined labelled dataset (Section 2.4) was prepared. All steps were executed inside Jupyter Notebook.

3.1 Fuzzy Similarity Feature Generation.

Run the notebook cell that produces fuzzy string similarity scores for each name pair using rapidfuzz. The following features are created:

- **TokenSetRatio**
- **PartialRatio**

These values quantify the surface-level similarity between the two names in each pair.

3.2 SBERT Semantic Similarity

Load the Sentence-BERT model (all-MiniLM-L6-v2) in the notebook and execute the encoding cell to generate semantic embeddings for:

- CustomerName
- SanctionName

Cosine similarity is computed using the SentenceTransformers util.cos_sim function.

```
model = SentenceTransformer('all-MiniLM-L6-v2')

emb1 = model.encode(pairs_df['CustomerName'].tolist(), convert_to_tensor=True)
emb2 = model.encode(pairs_df['SanctionName'].tolist(), convert_to_tensor=True)

pairs_df['SBERT_Similarity'] = util.cos_sim(emb1, emb2).diagonal().cpu().numpy()
```

Figure 9: Computing Name Similarities

3.3 Final Feature Dataset

Once all similarity scores are computed, the following DataFrame is created:

	TokenSetRatio	PartialRatio	SBERT_Similarity	Label
0	0.000000	0.000000	0.116736	1
1	18.181818	38.461538	-0.033618	1
2	40.000000	54.545455	0.115509	1
3	23.255814	42.105263	0.226910	1
4	21.621622	35.294118	0.129716	1

Figure 10: Final DataFrame

4 Model Training Setup

This section outlines the steps required to train the machine-learning models after the feature dataset is prepared.

4.1 Train Test Split

Run the cell that divides the feature dataset into training and testing sets:

```
# Core ML imports
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, roc_auc_score

from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC

# Split data
X = feature_df[['TokenSetRatio', 'PartialRatio', 'SBERT_Similarity']]
y = feature_df['Label']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

print(f"Train size: {X_train.shape}, Test size: {X_test.shape}")
```

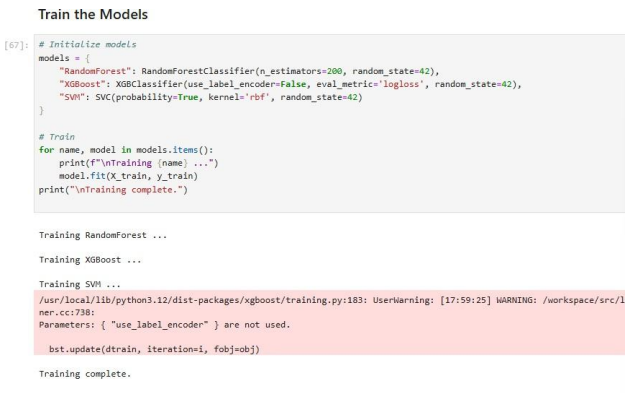
```
Train size: (1600, 3), Test size: (400, 3)
```

Figure 11: Splitting the data for training

4.2 Training the Models

Execute the model-training cells for:

- **Random Forest**
- **XGBoost**
- **SVM (RBF kernel)**



Train the Models

```
[67]: # Initialize models
models = {
    "RandomForest": RandomForestClassifier(n_estimators=200, random_state=42),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42),
    "SVM": SVC(probability=True, kernel='rbf', random_state=42)
}

# Train
for name, model in models.items():
    print(f"\nTraining {name} ...")
    model.fit(X_train, y_train)
print("\nTraining complete.")
```

```
Training RandomForest ...

Training XGBoost ...

Training SVM ...
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:183: UserWarning: [17:59:25] WARNING: /workspace/src/l
ner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)

Training complete.
```

Figure 12: Model training

4.3 Hyperparameter Tuning

Run the GridSearchCV blocks to perform tuning for each model

```python
best_models = {}

for name in models.keys():
    print(f"\nTuning and training {name} ...")

    grid = GridSearchCV(
        estimator=models[name],
        param_grid=param_grids[name],
        scoring='roc_auc',
        cv=5,
        n_jobs=-1
    )
    grid.fit(X_train, y_train)
    best_models[name] = grid.best_estimator_

    print(f"Best parameters for {name}: {grid.best_params_}")
    print(f"CV ROC-AUC: {grid.best_score_:.4f}")
```

Figure 13: Hyperparameter Tuning

4.4 Metrics

Some of the measures of performance that led to model behaviour were:

- Accuracy to define the accuracy of predicted matches.
- Recall to help capture true sanctioned entities.
- F1-score to balance recall and precision.
- ROC-AUC measure of discriminative ability.

5. Evaluation Setup

After model training, run the notebook cells that compute:

- classification metrics
- ROC curves
- confusion matrices

These commands evaluate model performance on the test set.

References

Gandhi, H., Tandon, K., Gite, S., Pradhan, B., & Alamri, A. (2024). Navigating the complexity of money laundering: anti–money laundering advancements with AI/ML insights. *International Journal on Smart Sensing and Intelligent Systems*, (1). https://sciendo.com/pdf/10.2478/ijssis-2024-0024

Kalusivalingam, A.K., Sharma, A., Patel, N. & Singh, V., (2022). Enhancing corporate governance and compliance through AI: Implementing natural language processing and machine learning algorithms. *International Journal of AI and ML*, *3*(9). https://cognitivecomputingjournal.com/index.php/IJAIML-V1/article/view/73

Kute, D.V., Pradhan, B., Shukla, N. & Alamri, A., (2021). Deep Learning and Explainable Artificial Intelligence Techniques Applied to Detecting Money Laundering: A Critical Review. *IEEE Access*, *9*, pp.82300-82317. https://ieeexplore.ieee.org/iel7/6287639/9312710/09446887.pdf

Pettersson Ruiz, E. & Angelis, J. (2022). Combating money laundering with machine learning–applicability of supervised-learning algorithms at cryptocurrency exchanges. *Journal of Money Laundering Control*, *25*(4), pp.766-778. https://www.emerald.com/insight/content/doi/10.1108/jmlc-09-2021-0106/full/pdf