National College of Ireland

Configuration Manual

MSc Research Project
MSCDAD_A

Ramanaidu Nallajonnala
Student ID: x23304197

School of Computing
National College of Ireland

Supervisor: Rejwanul Haque

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Ramanaidu Nallajonnala
Student ID:	x23304197
Programme:	MSCDAD_A **Year:** 2025
Module:	MSc Research Project
Lecturer:	Rejwanul Haque
Submission Due Date:	15/09/2025
Project Title:	AI-Enhanced Mental Health Risk Analysis Using Social Media Data

Word Count: 993 **Page Count:** 12

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: Ramanaidu Nallajonnala

Date: 15/09/2025

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

Ramanaidu Nallajonnala
Student ID: x23304197

1 Introduction

This configuration manual outlines the step-by-step process for setting up, executing, and evaluating an AI-based pipeline for detecting mental health risk indicators from Twitter data. The pipeline integrates data preprocessing, feature engineering, exploratory data analysis (EDA), classical machine learning models, recurrent neural networks (LSTM), and transformer-based models (DistilBERT). It also incorporates explainability techniques using SHAP to ensure transparency in predictions.
The implementation is conducted in Python 3.12, leveraging libraries such as Pandas, NumPy, Scikit-learn, Matplotlib, Seaborn, TensorFlow/Keras, Hugging Face Transformers, and TextBlob.

2 Dataset Loading and Basic Inspection

1. **Importing Libraries**

The process begins with importing essential Python libraries for data manipulation, visualization, and machine learning.

2. **Loading the Dataset**
   - The dataset Mental-Health-Twitter.csv is read using pandas.read_csv() into a DataFrame named df.
   - The dataset contains 20,000 entries with features such as tweet content, timestamps, engagement metrics (followers, retweets, favourites), and a binary label (0 = Non-Depressed, 1 = Depressed).

```
# Load the uploaded dataset
file_path = "Mental-Health-Twitter.csv"
df = pd.read_csv(file_path)

# Basic info and structure
basic_info = df.info()
head = df.head()
description = df.describe(include='all')
null_values = df.isnull().sum()

head, description, null_values
```

3. **Initial Data Inspection**
   - df.info() is used to inspect the structure, data types, and null values.
   - df.head() previews the first five records.
   - df.describe(include='all') provides statistical summaries for numeric and categorical features.
   - df.isnull().sum() confirms that there are no missing values in any column.

3 Data Cleaning and Preprocessing

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Check for null values
print("Null values in each column:\n")
print(df.isnull().sum())
```

```
Null values in each column:

Unnamed: 0      0
post_id         0
post_created    0
post_text       0
user_id         0
followers       0
friends         0
favourites      0
statuses        0
retweets        0
label           0
dtype: int64
```

```
# Drop unnecessary column
df.drop(columns=["Unnamed: 0"], inplace=True)

# Convert post_created to datetime
df['post_created'] = pd.to_datetime(df['post_created'], errors='coerce')

# Add post length feature
df['text_length'] = df['post_text'].apply(len)
```

1. **Removing Unnecessary Columns**
   - The Unnamed: 0 index column is dropped as it is redundant.
2. **Datetime Conversion**
   - The post_created column is converted to a datetime object for temporal feature extraction.
3. **Feature Creation: Text Length**
   - A text_length feature is computed by applying Python's len() function to each tweet.
   -

4 Exploratory Data Analysis (EDA)

EDA is performed to identify feature distributions, relationships, and potential data patterns.

1. **Label Distribution**
   - A count plot shows the number of depressed vs. non-depressed tweets.

```
# Plot: Distribution of Labels
plt.figure(figsize=(6,4))
sns.countplot(data=df, x='label')
plt.title("Distribution of Mental Health Labels (0 = Non-Depressed, 1 = Depressed)")
plt.xlabel("Label")
plt.ylabel("Count")
plt.tight_layout()
plt.show()
```

2. **Text Length by Label**
   - A boxplot compares tweet length across the two classes.

```
# Plot: Text Length by Label
plt.figure(figsize=(8,5))
sns.boxplot(data=df, x='label', y='text_length')
plt.title("Tweet Text Length Distribution by Label")
plt.xlabel("Label")
plt.ylabel("Tweet Text Length")
plt.tight_layout()
plt.show()
```

3. **Retweets Distribution**
   - A violin plot visualizes retweet counts per label on a logarithmic scale.

```
# Plot: Retweets vs Label
plt.figure(figsize=(8,5))
sns.violinplot(data=df, x='label', y='retweets', scale='width', inner='quartile')
plt.yscale('log')
plt.title("Distribution of Retweets by Label (Log Scale)")
plt.xlabel("Label")
plt.ylabel("Retweets")
plt.tight_layout()
plt.show()
```

4. **Correlation Heatmap**
   - A correlation matrix examines relationships between numeric features such as followers, friends, favourites, statuses, retweets, and text length.

```
# Correlation heatmap
numeric_df = df[['followers', 'friends', 'favourites', 'statuses', 'retweets', 'text_length', 'label']]
plt.figure(figsize=(10,6))
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Matrix of Features")
plt.tight_layout()
plt.show()
```

5 Feature Engineering

To enhance model predictive power, multiple feature transformations are performed:

1. **Sentiment Analysis**
   - Using **TextBlob**, a sentiment polarity score is computed for each tweet (-1 = negative, +1 = positive).
2. **Sentiment Shift (Trajectory)**
   - Tweets are sorted by user_id and post_created.
   - The sentiment change between consecutive tweets per user is computed using .diff() and stored as sentiment_shift.
3. **Temporal Posting Patterns**
   - hour: Extracted posting hour from post_created.
   - weekday: Extracted posting day name from post_created.

4. **Social Network Centrality**
    - **NetworkX** is used to simulate user retweet relationships and compute degree centrality as a measure of social influence.
5. **Final Feature Set**
    - The dataset is reduced to essential features: text_length, followers, friends, favourites, statuses, retweets, sentiment, sentiment_shift, hour, weekday, centrality, and label.

```
from textblob import TextBlob
import networkx as nx

# 1. Sentiment Analysis using TextBlob
df['sentiment'] = df['post_text'].apply(lambda x: TextBlob(str(x)).sentiment.polarity)

# 2. Sentiment Shift (trajectory) per user
df = df.sort_values(by=['user_id', 'post_created'])
df['sentiment_shift'] = df.groupby('user_id')['sentiment'].diff().fillna(0)

# 3. Temporal Posting Features
df['hour'] = df['post_created'].dt.hour
df['weekday'] = df['post_created'].dt.day_name()

# 4. Social Network Centrality (Dummy using retweets)
# Since actual retweet targets are unavailable, use a placeholder target
retweet_df = df[df['retweets'] > 0][['user_id']].copy()
retweet_df['target'] = 'retweeted_user'  # Placeholder
edges = retweet_df[['user_id', 'target']]

# Build graph and calculate degree centrality
G = nx.from_pandas_edgelist(edges, source='user_id', target='target', create_using=nx.DiGraph())
centrality_dict = nx.degree_centrality(G)
df['centrality'] = df['user_id'].map(centrality_dict).fillna(0)

# 5. Final Feature Selection
df_final = df[['user_id', 'post_created', 'text_length', 'followers', 'friends',
               'favourites', 'statuses', 'retweets', 'sentiment', 'sentiment_shift',
               'hour', 'weekday', 'centrality', 'label']]
```

6 Logistic Regression Baseline Model

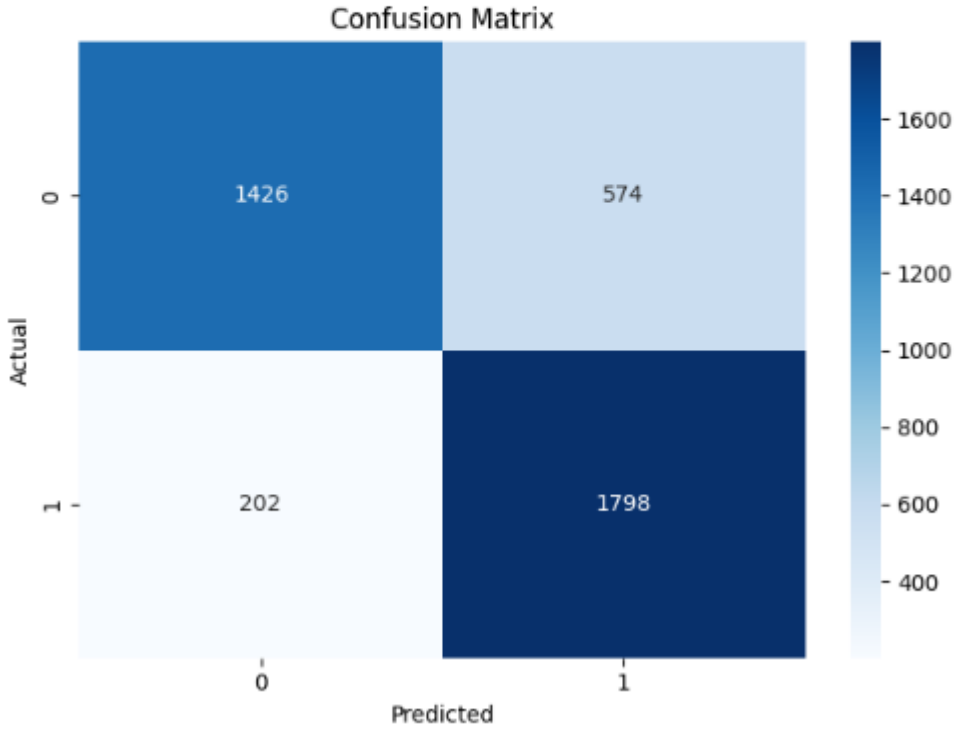
1. **Encoding and Scaling**
    - weekday is label-encoded to a numeric value.
    - Features are standardized using StandardScaler().

```python
# 1. Encode 'weekday' feature
le = LabelEncoder()
df_final['weekday_encoded'] = le.fit_transform(df_final['weekday'])

# 2. Define feature columns and target
features = ['text_length', 'followers', 'friends', 'favourites',
            'statuses', 'retweets', 'sentiment', 'sentiment_shift',
            'hour', 'weekday_encoded', 'centrality']
target = 'label'

X = df_final[features]
y = df_final[target]

# 3. 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
)

# 4. Feature Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 5. Baseline Model: Logistic Regression
model = LogisticRegression(max_iter=1000)
model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled)

# 6. Evaluation
print("\n Classification Report:\n")
print(classification_report(y_test, y_pred))

print("\n Confusion Matrix:")
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.tight_layout()
plt.show()

print("\n Accuracy Score:", accuracy_score(y_test, y_pred))
```

2. **Data Splitting**
    - The dataset is split into training (80%) and testing (20%) sets with stratification on labels.
3. **Model Training**
    - A **Logistic Regression** model is trained with max_iter=1000.
4. **Model Evaluation**

```
Classification Report:

              precision    recall  f1-score   support

           0       0.88      0.71      0.79      2000
           1       0.76      0.90      0.82      2000

    accuracy                           0.81      4000
   macro avg       0.82      0.81      0.80      4000
weighted avg       0.82      0.81      0.80      4000
```

```
Confusion Matrix:
```




```
Accuracy Score: 0.806
```

- Metrics: Accuracy (0.806), precision, recall, F1-score, and confusion matrix are generated.
- SHAP analysis identifies top predictive features (e.g., sentiment, text_length, retweets).

7 SHAP Explainability

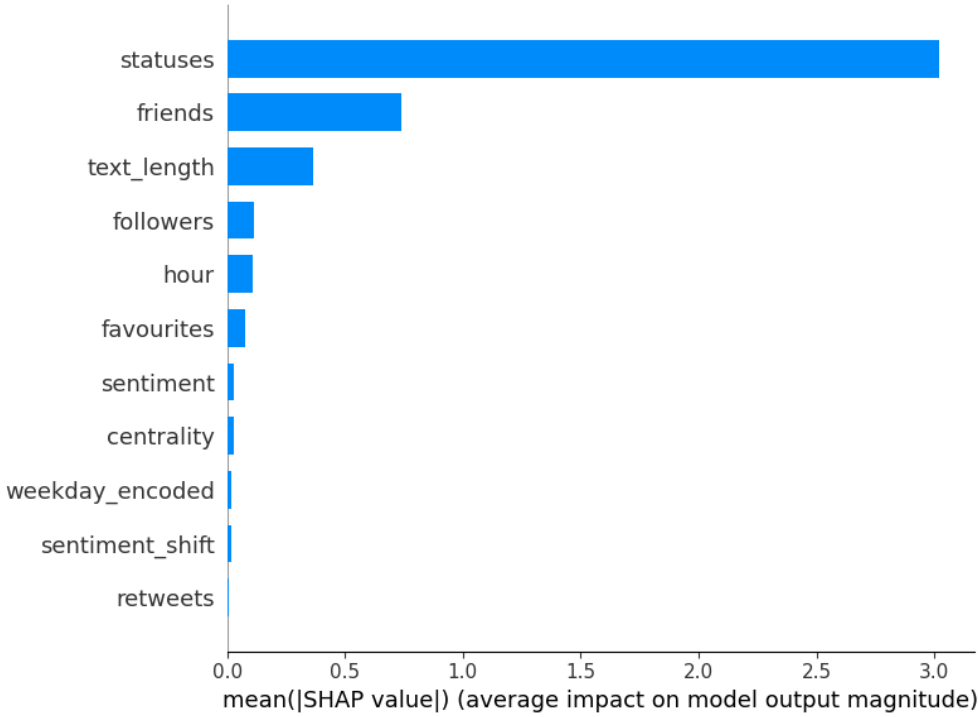
The SHAP framework is integrated to provide feature-level interpretability:

1. **Explainer Setup**
   - shap.LinearExplainer() is used for Logistic Regression models.

```python
print("\n Generating SHAP values (LinearExplainer)...")

# Select 100 test samples
X_sample = X_test_scaled[:100]
X_display = pd.DataFrame(X_sample, columns=features)

# Use LinearExplainer (designed for linear models like LogisticRegression)
explainer = shap.LinearExplainer(model, X_train_scaled, feature_names=features)

# Compute SHAP values for the selected test set
shap_values = explainer.shap_values(X_sample)

# SHAP summary plot (dot)
print("\n SHAP Summary Plot for Class 1 (Depressed):")
shap.summary_plot(shap_values, X_display)

# SHAP bar plot (importance)
print("\n SHAP Feature Importance (Bar Plot):")
shap.summary_plot(shap_values, X_display, plot_type="bar")
```

```
Generating SHAP values (LinearExplainer)...

SHAP Summary Plot for Class 1 (Depressed):
```

2. **Importance Visualization**
   - SHAP summary dot plots show feature contributions for depressed class predictions.
   - Bar plots rank features by global importance.



8 LSTM Temporal Sentiment Model

To incorporate sequential emotional patterns:

1. **Data Preparation**
   - Sentiment scores per user are aggregated into lists.
   - Sequences are padded to a fixed length (10 timesteps).
2. **Train-Test Split**

- The data is split with stratification.

3. **Model Architecture**
   - **LSTM** layer with 64 units.
   - **Dropout** (0.3) for regularization.
   - **Dense** output layer with sigmoid activation.
4. **Training**
   - Optimizer: Adam, Loss: Binary Crossentropy.
   - 10 epochs, batch size 32.

```
# Load processed dataset
df = pd.read_csv("processed_mental_health_dataset.csv")
df['post_created'] = pd.to_datetime(df['post_created'])
df = df.sort_values(by=['user_id', 'post_created'])

# Prepare sentiment trajectory data per user
user_sentiments = df.groupby('user_id')['sentiment'].apply(list)
labels = df.groupby('user_id')['label'].first()

# Pad sequences to equal length
def pad_sentiment(seq, maxlen=10):
    return [0.0] * (maxlen - len(seq)) + seq[-maxlen:]

X_seq = np.array([pad_sentiment(seq) for seq in user_sentiments])
y = labels.values

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

# LSTM model
model = Sequential()
model.add(LSTM(64, input_shape=(X_train.shape[1], 1)))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train
model.fit(X_train[..., np.newaxis], y_train, epochs=10, batch_size=32, validation_split=0.2)

# Evaluate
y_pred = (model.predict(X_test[..., np.newaxis]) > 0.5).astype(int)
print(classification_report(y_test, y_pred))
print("AUC Score:", roc_auc_score(y_test, y_pred))
```

5. Results

```
Epoch 1/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 3s 472ms/step - accuracy: 0.5771 - loss: 0.6923 - val_accuracy: 0.7500 - val_loss: 0.6864
Epoch 2/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 115ms/step - accuracy: 0.7685 - loss: 0.6847 - val_accuracy: 0.7500 - val_loss: 0.6800
Epoch 3/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 123ms/step - accuracy: 0.7433 - loss: 0.6789 - val_accuracy: 0.7500 - val_loss: 0.6733
Epoch 4/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 110ms/step - accuracy: 0.7433 - loss: 0.6728 - val_accuracy: 0.7500 - val_loss: 0.6660
Epoch 5/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 132ms/step - accuracy: 0.7285 - loss: 0.6655 - val_accuracy: 0.7500 - val_loss: 0.6578
Epoch 6/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 115ms/step - accuracy: 0.7285 - loss: 0.6545 - val_accuracy: 0.7500 - val_loss: 0.6484
Epoch 7/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 120ms/step - accuracy: 0.7225 - loss: 0.6498 - val_accuracy: 0.7500 - val_loss: 0.6378
Epoch 8/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 108ms/step - accuracy: 0.7641 - loss: 0.6325 - val_accuracy: 0.7500 - val_loss: 0.6256
Epoch 9/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 177ms/step - accuracy: 0.7329 - loss: 0.6272 - val_accuracy: 0.7500 - val_loss: 0.6120
Epoch 10/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 133ms/step - accuracy: 0.7537 - loss: 0.6055 - val_accuracy: 0.7500 - val_loss: 0.5972
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 233ms/step
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         4
           1       0.73      1.00      0.85        11

    accuracy                           0.73        15
   macro avg       0.37      0.50      0.42        15
weighted avg       0.54      0.73      0.62        15

AUC Score: 0.5
```

- Accuracy: 73%, but poor performance on non-depressed class due to data sparsity.
- AUC: 0.50, indicating chance-level separation.

9 Transformer-Based Model: DistilBERT

For deep contextual text understanding:

```
# Tokenizer
tokenizer = DistilBertTokenizer.from_pretrained("distilbert-base-uncased")

# Tokenize text
inputs = tokenizer(
    list(df['post_text']),
    padding=True,
    truncation=True,
    return_tensors='tf',
    max_length=64
)

# Labels
labels = df['label'].values

# === Step 3: Train-test split (stratified) ===
X_train_idx, X_test_idx, y_train, y_test = train_test_split(
    np.arange(len(labels)),
    labels,
    test_size=0.2,
    stratify=labels,
    random_state=42
)

X_train = {key: tf.gather(val, X_train_idx) for key, val in inputs.items()}
X_test = {key: tf.gather(val, X_test_idx) for key, val in inputs.items()}
X_train["labels"] = tf.convert_to_tensor(y_train)
X_test["labels"] = tf.convert_to_tensor(y_test)

# === Step 4: Compute class weights manually ===
class_weights = class_weight.compute_class_weight(
    class_weight='balanced',
    classes=np.unique(y_train),
    y=y_train
)
class_weights_tensor = tf.constant(class_weights, dtype=tf.float32)

# === Step 5: Load DistilBERT model ===
model = TFDistilBertForSequenceClassification.from_pretrained("distilbert-base-uncased", num_labels=2)
```

```python
# === Step 6: Optimizer ===
batch_size = 16
epochs = 3
steps_per_epoch = len(y_train) // batch_size
num_train_steps = steps_per_epoch * epochs
num_warmup_steps = int(0.1 * num_train_steps)

optimizer, _ = create_optimizer(
    init_lr=2e-5,
    num_train_steps=num_train_steps,
    num_warmup_steps=num_warmup_steps
)

# === Step 7: Custom Weighted Loss Function ===
def weighted_loss(y_true, y_pred):
    y_true = tf.cast(y_true, tf.int32)
    weights = tf.gather(class_weights_tensor, y_true)
    loss = tf.keras.losses.sparse_categorical_crossentropy(y_true, y_pred, from_logits=True)
    return loss * weights

# === Step 8: Compile model ===
model.compile(
    optimizer=optimizer,
    loss=weighted_loss,
    metrics=['accuracy']
)

# === Step 9: Train model ===
history = model.fit(
    X_train,
    validation_split=0.1,
    batch_size=batch_size,
    epochs=epochs
)
```

1. **Tokenizer Initialization**
   - **DistilBertTokenizer** is loaded from Hugging Face (distilbert-base-uncased).
2. **Text Tokenization**
   - Tweets are tokenized with padding, truncation (max length 64), and converted to tensors.
3. **Data Splitting**
   - Stratified train-test split.
4. **Class Weights**
   - Computed to address label imbalance.
5. **Model Initialization**
   - **TFDistilBertForSequenceClassification** with num_labels=2.
6. **Optimizer and Learning Rate Scheduling**
   - AdamW optimizer with warmup steps (10% of training steps).
   - Initial learning rate: 2e-5.
7. **Custom Weighted Loss Function**
   - Sparse categorical crossentropy with class-weight adjustments.
8. **Training**
   - Batch size: 16, Epochs: 3, Validation split: 0.1.
9. **Performance**

```python
# === Predict logits on test set ===
logits = model.predict(X_test).logits
probs = tf.nn.softmax(logits, axis=-1)
y_pred = tf.argmax(probs, axis=1).numpy()

# === Classification report ===
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
125/125 [==============================] - 334s 3s/step

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.90      0.91      2000
           1       0.91      0.92      0.91      2000

    accuracy                           0.91      4000
   macro avg       0.91      0.91      0.91      4000
weighted avg       0.91      0.91      0.91      4000
```

```python
# === AUC Score ===
try:
    print("AUC Score:", roc_auc_score(y_test, probs[:, 1]))
except ValueError as e:
    print("AUC could not be computed:", e)
```

```
AUC Score: 0.975700875
```

- Accuracy: 91%, balanced precision/recall for both classes.
- AUC: 0.975, indicating excellent discriminative ability.

10 Hardware and Software Requirements

- **Python Version**: 3.12
- **Libraries**:
  - Data Processing: pandas, numpy
  - Visualization: matplotlib, seaborn
  - NLP: textblob, transformers
  - Graph Analysis: networkx
  - ML/DL: scikit-learn, tensorflow, keras
  - Explainability: shap
- **Hardware**:
  - GPU recommended for DistilBERT training.
  - Minimum 16 GB RAM for smooth execution.

References

Baydili, İ., Tasci, B., & Tasci, G. (2025). Deep learning-based detection of depression and suicidal tendencies in social media data with feature selection. *Behavioral Sciences*, *15*(3), 352. https://www.mdpi.com/2076-328X/15/3/352

Gogula, G. (2024). *Mental Illness Detection Using NLP* (Doctoral dissertation, CALIFORNIA STATE UNIVERSITY, NORTHRIDGE). https://scholarworks.calstate.edu/downloads/ms35th72p

Kashif, A., Hassan, S., Dad, A.M., Malik, M.H., Rana, M., Rehman, S.U., Allahham, M., Verschueren, R. and Ness, S., 2024. Examining the Impact of AI-Enhanced Social Media Content on Adolescent Well-being in the Digital Age. *Kurdish Studies*, *12*(2), pp.771-789. https://www.researchgate.net/profile/Saima-Hassan-5/publication/378204677_Examining_the_Impact_of_AI-Enhanced_Social_Media_Content_on_Adolescent_Well-being_in_the_Digital_Age/links/663935c908aa54017ae2ac56/Examining-the-Impact-of-AI-Enhanced-Social-Media-Content-on-Adolescent-Well-being-in-the-Digital-Age.pdf

Kurniadi, F. I., Paramita, N. L. P. S. P., Sihotang, E. F. A., Anggreainy, M. S., & Zhang, R. (2024). BERT and RoBERTa Models for Enhanced Detection of Depression in Social Media Text. *Procedia Computer Science*, *245*, 202-209.