Phase - 3 Solution Development and Testing

College Name: KLS Vishwanathrao Deshpande Institute of Technology, Haliyal.

Group Members:

- Name: SHRAVAN PATIL
  Number: CAN_32896536

- Name: SALMA B. NADAF
  CAN ID Number:
  CAN_32858276

- Name: AKASH MALAKAIGOL
  CAN ID Number:
  CAN_32906021

- Name: MEGHARAJ
  KSHATRIYA. CAN ID Number:
  CAN_ 34001056

Project Title: AI-Powered Duplicate Data Detection

Model development and evaluation

Step 1: Advanced Data Cleaning

```
import pandas as pd
import numpy as np
from sklearn.impute import KNNImputer
from sklearn.ensemble import IsolationForest, RandomForestClassifier
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score
from flask import Flask, request, jsonify
import joblib
import dash
import dash_core_components as dcc
import dash_html_components as html
import plotly.express as px
```

Step 2: Building of Training Models

```
# Handling Missing Values
imputer = KNNImputer(n_neighbors=5)
data_cleaned = pd.DataFrame(imputer.fit_transform(data), columns=data.columns)

# Outlier Detection
iso = IsolationForest(contamination=0.01, random_state=42)
data_cleaned['Anomaly'] = iso.fit_predict(data_cleaned.drop(columns=['target']))
data_cleaned = data_cleaned[data_cleaned['Anomaly'] == 1].drop(columns=['Anomaly'])

# Addressing Imbalanced Classes
smote = SMOTE(random_state=42, k_neighbors=5, sampling_strategy='minority')
X_resampled, y_resampled = smote.fit_resample(data_cleaned.drop(columns=['target']),
data_cleaned['target'])
```

Step 3: Exploratory Data Analysis (EDA)

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

# Load dataset
df = pd.read_csv("duplicate_data.csv")

# Step 1: Data Visualization
plt.figure(figsize=(12, 6))
df.hist(figsize=(12, 6), bins=30)
plt.suptitle("Feature Distributions", fontsize=15)
plt.show()

sns.pairplot(df, hue="is_duplicate", diag_kind="kde")
plt.suptitle("Scatter Plot of Duplicate vs Non-Duplicate Records", fontsize=15)
plt.show()

plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Feature Correlation Heatmap")
plt.show()

# Step 2: Data Quality Checks
print("Missing Values:\n", df.isnull().sum())

plt.figure(figsize=(12, 6))
sns.boxplot(data=df, palette="coolwarm")
plt.title("Outlier Detection Using Boxplot")
plt.xticks(rotation=45)
plt.show()

# Step 3: Correlation Analysis
correlation_matrix = df.corr()
high_corr_features = correlation_matrix[abs(correlation_matrix) > 0.5]
print("Highly Correlated Features:\n", high_corr_features)

# Step 4: Pattern Detection using DBSCAN
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df.drop(columns=["is_duplicate"]))
dbscan = DBSCAN(eps=0.5, min_samples=5).fit(df_scaled)
df["cluster"] = dbscan.labels_

plt.figure(figsize=(10, 6))
sns.scatterplot(x=df.iloc[:, 0], y=df.iloc[:, 1], hue=df["cluster"], palette="viridis")
plt.title("Clustered Duplicate Data Detection")
plt.show()

print("Pattern Detection Complete: Clusters Identified")
```

Step 4: Model Evaluation

```
metrics = {
    "Decision Tree": {
        "Accuracy": accuracy_score(y_test, y_pred_dt),
        "Precision": precision_score(y_test, y_pred_dt),
        "Recall": recall_score(y_test, y_pred_dt),
        "F1 Score": f1_score(y_test, y_pred_dt),
        "ROC AUC": roc_auc_score(y_test, y_pred_dt)
    },
    "Random Forest": {
        "Accuracy": accuracy_score(y_test, y_pred_rf),
        "Precision": precision_score(y_test, y_pred_rf),
        "Recall": recall_score(y_test, y_pred_rf),
        "F1 Score": f1_score(y_test, y_pred_rf),
        "ROC AUC": roc_auc_score(y_test, y_pred_rf)
    }
}
```

Step 5: Results and Insights

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix
from fuzzywuzzy import fuzz
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import RandomForestClassifier
import joblib
from flask import Flask, request, jsonify
import dash
import dash_core_components as dcc
import dash_html_components as html
import plotly.express as px

# Step 1: Data Preprocessing
def preprocess_data(df):
    df = df.drop_duplicates()
    df = df.dropna()
    return df

# Step 2: Feature Engineering
def calculate_similarity(row):
    return fuzz.token_sort_ratio(row['col1'], row['col2'])

def transform_data(df):
    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform(df['col1'])
    return tfidf_matrix

# Step 3: Model Training
def train_model(X, y):
    X_resampled, y_resampled = SMOTE().fit_resample(X, y)
    X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2)
    model = RandomForestClassifier(n_estimators=50, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(f"Precision: {precision_score(y_test, y_pred):.2f}")
    print(f"Recall: {recall_score(y_test, y_pred):.2f}")
    print(f"F1-score: {f1_score(y_test, y_pred):.2f}")
    print(confusion_matrix(y_test, y_pred))
    joblib.dump(model, "duplicate_detector.pkl")
    return model

# Step 4: Deployment with Flask
app = Flask(__name__)
model = joblib.load("duplicate_detector.pkl")

@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json()
    df = pd.DataFrame([data])
    prediction = model.predict(df)
    return jsonify({"prediction": prediction.tolist()})

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=5000, debug=True)

# Step 5: Monitoring Dashboard with Dash
dash_app = dash.Dash(__name__)
df = pd.read_csv("data.csv")
fig = px.histogram(df, x="similarity_score")
dash_app.layout = html.Div([dcc.Graph(figure=fig)])

dash_app.run_server(debug=True)
```

Step 6:Deployment and Integration

```
import tensorflow as tf
import pandas as pd
import numpy as np
import requests
import os
from flask import Flask, request, jsonify
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# ------------------------
# 1️⃣ Model Training & Saving
# ------------------------
```

```
print("🚀 Training AI model for duplicate detection...")

# Generate sample data
data = pd.DataFrame({
    'feature1': np.random.rand(1000),
    'feature2': np.random.rand(1000),
    'duplicate': np.random.choice([0, 1], size=1000)  # 0 = Not Duplicate, 1 = Duplicate
})

X = data[['feature1', 'feature2']]
y = data['duplicate']

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build AI model
model = Sequential([
    Dense(16, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(8, activation='relu'),
    Dense(1, activation='sigmoid')
])

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

# Train model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))

# Save model for TensorFlow Serving
MODEL_DIR = "duplicate_detection_model"
tf.saved_model.save(model, MODEL_DIR)

print(f"✅ Model trained & saved at '{MODEL_DIR}'!")

# ------------------------
# 2️⃣ Flask API for Predictions
# ------------------------

app = Flask(__name__)
TF_SERVING_URL = "http://localhost:8501/v1/models/duplicate_detection_model:predict"

@app.route('/predict', methods=['POST'])
def predict():
    try:
        input_data = request.get_json()
        payload = {"instances": [input_data]}
        response = requests.post(TF_SERVING_URL, json=payload)
        prediction = response.json()
        return jsonify(prediction)
    except Exception as e:
        return jsonify({"error": str(e)}), 500

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)

# ------------------------
# 3️⃣ Docker Instructions
# ------------------------
dockerfile_content = """
# Use Python base image
FROM python:3.9

# Set working directory
WORKDIR /app

# Copy script & install dependencies
COPY requirements.txt .
COPY api.py .

# Install dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Expose Flask API port
EXPOSE 5000

# Start Flask API
CMD ["python", "api.py"]
"""

with open("Dockerfile", "w") as f:
    f.write(dockerfile_content)

requirements_content = """
flask
requests
tensorflow
pandas
numpy
scikit-learn
"""

with open("requirements.txt", "w") as f:
    f.write(requirements_content)
```

```
# ------------------------
# 4️⃣ TensorFlow Serving Script
# ------------------------

tf_serving_script = """
#!/bin/bash
docker run -p 8501:8501 --name=tf_serving \\
  --mount type=bind,source=$(pwd)/duplicate_detection_model,target=/models/duplicate_detection_model \\
  -e MODEL_NAME=duplicate_detection_model -t tensorflow/serving
"""

with open("run_tf_serving.sh", "w") as f:
    f.write(tf_serving_script)

print("✅ Flask API & Docker setup complete! 🚀")
```