```
diabetes_codes = {
  "Regular insulin dose": 33,
  "NPH insulin dose": 34,
  "UltraLente insulin dose": 35,
  "Unspecified blood glucose measurement": 48,
  "Pre-breakfast blood glucose measurement": 58,
  "Post-breakfast blood glucose measurement": 59,
  "Pre-lunch blood glucose measurement": 60,
  "Post-lunch blood glucose measurement": 61,
  "Pre-supper blood glucose measurement": 62,
  "Post-supper blood glucose measurement": 63,
  "Pre-snack blood glucose measurement": 64,
  "Hypoglycemic symptoms": 65,
  "Typical meal ingestion": 66,
  "More-than-usual meal ingestion": 67,
  "Less-than-usual meal ingestion": 68,
  "Typical exercise activity": 69,
  "More-than-usual exercise activity": 70,
  "Less-than-usual exercise activity": 71,
  "Unspecified special event": 72
}
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
from sklearn.preprocessing import StandardScaler

df_diabetes = pd.read_csv("diabetes-clean.csv")
df_heartdis = pd.read_csv("heartdis-clean.csv")
df_thyroid = pd.read_csv("thyroid-clean.csv")
```

```
df_diabetes.info()


df_heartdis.info()


df_thyroid.info()
from sklearn.ensemble import RandomForestClassifier


# Separate features and target
X = df_diabetes[['code']]
y = df_diabetes['diabetes_type']


# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Initialize and train the SVM model
diabetes_model = RandomForestClassifier(random_state=42)
diabetes_model.fit(X_train, y_train)


# Make predictions
y_pred = diabetes_model.predict(X_test)
prob = diabetes_model.predict_proba(X_test)


# Calculate metrics
f1 = f1_score(y_test, y_pred, average='weighted')
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')


# Print metrics
print("F1 Score:", f1)
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)

from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC

X = df_heartdis.drop('heartdis_type', axis=1)
y = df_heartdis['heartdis_type']

categorical_cols = X.select_dtypes(include=['object']).columns.tolist()
numerical_cols = X.select_dtypes(include=['int64', 'float64']).columns.tolist()
bool_cols = X.select_dtypes(include=['bool']).columns.tolist()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=12)

preprocessor = ColumnTransformer(
    transformers=[
        ('num', 'passthrough', numerical_cols),
        ('cat', OneHotEncoder(), categorical_cols),
        ('scaler', StandardScaler(), numerical_cols)
    ])

heartdis_model_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('model', SVC(probability=True, kernel='rbf', C=10000, random_state=42))
])

heartdis_model_pipeline.fit(X_train, y_train)
```

```
y_pred = heartdis_model_pipeline.predict(X_test)
prob = heartdis_model_pipeline.predict_proba(X_test)

f1 = f1_score(y_test, y_pred, average='weighted')
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')

print("F1 Score:", f1)
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)

from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC

X = df_thyroid.drop('has_thyroid_disease', axis=1)
y = df_thyroid['has_thyroid_disease']

categorical_cols = ['sex']
bool_cols = X.select_dtypes(include=['bool']).columns.tolist()
numerical_cols = X.select_dtypes(include=['int64', 'float64']).columns.tolist()

preprocessor = ColumnTransformer(
    transformers=[
        ('num', 'passthrough', numerical_cols),
        ('cat', OneHotEncoder(), categorical_cols),
        ('scaler', StandardScaler(), numerical_cols)
    ])
```

```
thyroid_model_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('svm', SVC(probability=True, kernel='rbf', C=10, random_state=42))
])

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

thyroid_model_pipeline.fit(X_train, y_train)

y_pred = thyroid_model_pipeline.predict(X_test)
prob = thyroid_model_pipeline.predict_proba(X_test)

f1 = f1_score(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

print("F1 Score:", f1)
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)

def get_user_data(str):
    """
    Arguments
    - str: 'diabetes', 'heartdis', or 'thyroid'
    """
    if str == "diabetes":
        X = df_diabetes.drop('diabetes_type', axis=1)
        y = df_diabetes['diabetes_type']
    elif str == "heartdis":
        X = df_heartdis.drop('heartdis_type', axis=1)
        y = df_heartdis['heartdis_type']
    elif str == "thyroid":
        X = df_thyroid.drop('has_thyroid_disease', axis=1)
        y = df_thyroid['has_thyroid_disease']
    else:
        raise ValueError("Invalid input: expected 'diabetes', 'heartdis', or 'thyroid'")
    _, X_test, _, _ = train_test_split(X, y, test_size=0.2, random_state=42)
    random_selection = X_test.sample(1)
    return random_selection
def get_dscore(row):
    classes = diabetes_model.classes_
    prediction = diabetes_model.predict_proba(row)
    probas = { classes[i]: prediction[0][i] for i in range(len(classes)) }
    return max(probas.values())


def get_hscore(row):
    classes = heartdis_model_pipeline.classes_
    prediction = heartdis_model_pipeline.predict_proba(row)
    probas = { classes[i]: prediction[0][i] for i in range(len(classes)) }
    return sum(probas[class_label] * int(class_label) for class_label in probas)


def get_tscore(row):
    classes = thyroid_model_pipeline.classes_
    prediction = thyroid_model_pipeline.predict_proba(row)
    probas = { classes[i]: prediction[0][i] for i in range(len(classes)) }
    wtscore = sum(probas[class_label] * int(class_label) for class_label in probas)
    return wtscore / (sum([ int(class_label) for class_label in probas ]))


def low(x):
    return max(0, min(1, (0.4 - x) / 0.4))

def moderate(x):
    return max(0, min((x - 0.3) / 0.4, (0.7 - x) / 0.4))

def high(x):
    return max(0, min((x - 0.6) / 0.4, 1))

def fuzzify(score):
    return {
        "low": low(score),
        "moderate": moderate(score),
        "high": high(score)
    }

def evaluate_rules(d_score, h_score, t_score):
    # Fuzzify each input
    d_fuzzy = fuzzify(d_score)
    h_fuzzy = fuzzify(h_score)
    t_fuzzy = fuzzify(t_score)

    # Initialize outputs for risk levels
    low_risk, moderate_risk, high_risk = 0, 0, 0

    # Rules
    # Rule 1: IF d_score is high OR h_score is high THEN risk is high
    high_risk = max(high_risk, max(d_fuzzy['high'], h_fuzzy['high']))
    # Rule 2: IF d_score is moderate AND h_score is moderate AND t_score is moderate THEN risk is moderate
    moderate_risk = max(moderate_risk, min(d_fuzzy['moderate'], h_fuzzy['moderate'], t_fuzzy['moderate']))
    # Rule 3: IF d_score is low AND h_score is low AND t_score is low THEN risk is low
    low_risk = max(low_risk, min(d_fuzzy['low'], h_fuzzy['low'], t_fuzzy['low']))
    # Rule 4: IF d_score is high OR t_score is high THEN risk is moderate
    moderate_risk = max(moderate_risk, max(d_fuzzy['high'], t_fuzzy['high']))

    return low_risk, moderate_risk, high_risk

def defuzzify(low_risk, moderate_risk, high_risk):
    risk_levels = {
        "low": 0.25,
        "moderate": 0.5,
        "high": 0.75
    }
    numerator = (low_risk * risk_levels["low"] +
                 moderate_risk * risk_levels["moderate"] +
                 high_risk * risk_levels["high"])
    denominator = low_risk + moderate_risk + high_risk
    return numerator / denominator if denominator != 0 else 0

def calculate_final_risk(d_score, h_score, t_score, threshold=0.5):
    low_risk, moderate_risk, high_risk = evaluate_rules(d_score, h_score, t_score)
    final_risk_score = defuzzify(low_risk, moderate_risk, high_risk)
    risk_status = "In Danger" if final_risk_score > threshold else "Not in Danger"
    return final_risk_score, risk_status

d_score = get_dscore(get_user_data('diabetes'))
h_score = get_hscore(get_user_data('heartdis'))
t_score = get_tscore(get_user_data('thyroid'))

print("Diabetes Score:", d_score)
print("Heart Disease Score:", h_score)
print("Thyroid Score:", t_score)
```

```
final_score, status = calculate_final_risk(d_score, h_score, t_score)

print("Final Risk Score:", final_score)

print("Risk Status:", status)
```