!gdown 1FGul56hum7JLk3WtotNjmzPs9-7yajOS  
!rm \*.csv  
!unzip \*.zip  
!rm \*.zip

Downloading...  
From: https://drive.google.com/uc?id=1FGul56hum7JLk3WtotNjmzPs9-7yajOS  
To: /content/ml-usi-disease-ds-clean.zip  
 0% 0.00/77.7k [00:00<?, ?B/s] 100% 77.7k/77.7k [00:00<00:00, 70.3MB/s]  
Archive: ml-usi-disease-ds-clean.zip  
 inflating: thyroid-clean.csv   
 inflating: diabetes-clean.csv   
 inflating: heartdis-clean.csv

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

# Loading Data

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()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 29257 entries, 0 to 29256  
Data columns (total 2 columns):  
 # Column Non-Null Count Dtype  
--- ------ -------------- -----  
 0 code 29257 non-null int64  
 1 diabetes\_type 29257 non-null int64  
dtypes: int64(2)  
memory usage: 457.3 KB

df\_heartdis.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 740 entries, 0 to 739  
Data columns (total 11 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 age 740 non-null int64   
 1 sex 740 non-null object   
 2 cp 740 non-null object   
 3 trestbps 740 non-null float64  
 4 chol 740 non-null float64  
 5 fbs 740 non-null bool   
 6 restecg 740 non-null object   
 7 thalch 740 non-null float64  
 8 exang 740 non-null bool   
 9 oldpeak 740 non-null float64  
 10 heartdis\_type 740 non-null int64   
dtypes: bool(2), float64(4), int64(2), object(3)  
memory usage: 53.6+ KB

df\_thyroid.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 3772 entries, 0 to 3771  
Data columns (total 21 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 age 3772 non-null int64   
 1 sex 3772 non-null object   
 2 on thyroxine 3772 non-null bool   
 3 query on thyroxine 3772 non-null bool   
 4 on antithyroid medication 3772 non-null bool   
 5 sick 3772 non-null bool   
 6 pregnant 3772 non-null bool   
 7 thyroid surgery 3772 non-null bool   
 8 I131 treatment 3772 non-null bool   
 9 query hypothyroid 3772 non-null bool   
 10 query hyperthyroid 3772 non-null bool   
 11 lithium 3772 non-null bool   
 12 goitre 3772 non-null bool   
 13 tumor 3772 non-null bool   
 14 psych 3772 non-null bool   
 15 TSH 3772 non-null float64  
 16 T3 3772 non-null float64  
 17 TT4 3772 non-null float64  
 18 T4U 3772 non-null float64  
 19 FTI 3772 non-null float64  
 20 has\_thyroid\_disease 3772 non-null bool   
dtypes: bool(14), float64(5), int64(1), object(1)  
memory usage: 258.0+ KB

# Making Model (SVM)

## Diabetes Model

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)

F1 Score: 0.7169263760694449  
Accuracy: 0.7445317840054683  
Precision: 0.7431638393945541  
Recall: 0.7445317840054683

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/\_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.  
 \_warn\_prf(average, modifier, f"{metric.capitalize()} is", len(result))

## Heart Disease Model

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)

F1 Score: 0.5237860468123625  
Accuracy: 0.5608108108108109  
Precision: 0.4966118533397394  
Recall: 0.5608108108108109

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/\_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.  
 \_warn\_prf(average, modifier, f"{metric.capitalize()} is", len(result))

## Thyroid Model

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)

F1 Score: 0.9738884968242767  
Accuracy: 0.9509933774834437  
Precision: 0.9583333333333334  
Recall: 0.9899569583931134

# Fuzzy Inference System

## Taking Input

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 ]))

## Evaluation

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)

Diabetes Score: 0.45546908340759445  
Heart Disease Score: 0.7107695324795081  
Thyroid Score: 0.6256012504488321  
Final Risk Score: 0.7030668338571375  
Risk Status: In Danger