**PROJECT TITLE: AI BASED DIABETES PREDICTION SYSTEM**

**Phase 4: Development Part 2**

In this phase, I continue building for my AI based Diabetes Prediction System project by performing different activities like feature engineering, model training, evaluation etc with the help of machine learning techniques.

**OBJECTIVE:**

The objective of the project to develop an AI-powered diabetes prediction system is to provide early risk assessment and personalized preventive measures for individuals, allowing them to take proactive actions to manage their health.

**DATASET:**

<https://www.kaggle.com/datasets/mathchi/diabetes-data-set>

**PROGRAM CODE FOR FEATURE ENGINEERING,MODEL TRAINING AND EVALUATION:**

*# Import necessary libraries*

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split, cross\_val\_score, GridSearchCV

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score

***# Step 1: Load the dataset***

data = pd.read\_csv('diabetes\_data.csv')

***# Step 2: Feature Engineering***

*# Categorize BMI as 'BMI\_Category'*

def categorize\_bmi(bmi):

if bmi < 18.5:

return 'Underweight'

elif 18.5 <= bmi < 24.9:

return 'Normal Weight'

elif 24.9 <= bmi < 29.9:

return 'Overweight'

else:

return 'Obese'

data['BMI\_Category'] = data['BMI'].apply(categorize\_bmi)

***# Step 3: Split the data into features (X) and target (y)***

X = data.drop('Outcome', axis=1)

y = data['Outcome']

***# Step 4: Split the dataset into training and testing sets***

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

***# Step 5: Feature Engineering and Preprocessing***

*# Apply feature scaling to numerical features*

numerical\_features = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']

categorical\_features = ['BMI\_Category']

*# Feature Scaling*

scaler = StandardScaler()

X\_train[numerical\_features] = scaler.fit\_transform(X\_train[numerical\_features])

X\_test[numerical\_features] = scaler.transform(X\_test[numerical\_features])

*# One-Hot Encoding for categorical features*

encoder = OneHotEncoder(drop='first', sparse=False)

X\_train\_encoded = encoder.fit\_transform(X\_train[categorical\_features])

X\_test\_encoded = encoder.transform(X\_test[categorical\_features])

*# Concatenate the encoded features with the original features*

X\_train = np.concatenate([X\_train.drop(categorical\_features, axis=1).values, X\_train\_encoded], axis=1)

X\_test = np.concatenate([X\_test.drop(categorical\_features, axis=1).values, X\_test\_encoded], axis=1)

***# Step 6: Model Training and Hyperparameter Tuning***

param\_grid = {

'n\_estimators': [50, 100, 150],

'max\_depth': [None, 10, 20, 30],

'min\_samples\_split': [2, 5, 10],

'min\_samples\_leaf': [1, 2, 4]

}

rf = RandomForestClassifier(random\_state=42)

grid\_search = GridSearchCV(estimator=rf, param\_grid=param\_grid, cv=5, scoring='accuracy')

grid\_search.fit(X\_train, y\_train)

*# Get the best model from the hyperparameter tuning*

best\_model = grid\_search.best\_estimator\_

***# Step 7: Model Evaluation with Cross-Validation***

cv\_scores = cross\_val\_score(best\_model, X\_train, y\_train, cv=5, scoring='accuracy')

print("Cross-Validation Scores:", cv\_scores)

print("Mean CV Accuracy:", np.mean(cv\_scores))

*# Evaluate the model on the test set*

y\_pred = best\_model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred)

recall = recall\_score(y\_test, y\_pred)

f1 = f1\_score(y\_test, y\_pred)

roc\_auc = roc\_auc\_score(y\_test, best\_model.predict\_proba(X\_test)[:, 1])

***# Step 8: Display Results***

print("Test Set Accuracy:", accuracy)

print("Test Set Precision:", precision)

print("Test Set Recall:", recall)

print("Test Set F1 Score:", f1)

print("Test Set ROC AUC Score:", roc\_auc)

**EXPLANATION FOR FEATURE ENGINEERING,MODEL TRAINING AND EVALUATION:**

The provided Python code performs feature engineering, model training and evaluation using machine learning techniques.

**Feature Engineering:**

def categorize\_bmi(bmi):

if bmi < 18.5:

return 'Underweight'

elif 18.5 <= bmi < 24.9:

return 'Normal Weight'

elif 24.9 <= bmi < 29.9:

return 'Overweight'

else:

return 'Obese'

data['BMI\_Category'] = data['BMI'].apply(categorize\_bmi)

* In this step, feature engineering is performed.
* A new feature, 'BMI\_Category,' is created based on the 'BMI' feature.
* The BMI values are categorized into four categories: 'Underweight,' 'Normal Weight,' 'Overweight,' and 'Obese.'
* This categorization allows the model to capture potentially meaningful patterns related to BMI ranges, which may be relevant to diabetes prediction.

Split the Data:

X = data.drop('Outcome', axis=1)

y = data['Outcome']

* Here, the dataset is split into feature set `X` (independent variables) and the target variable `y` (the 'Outcome' column), which represents whether a patient has diabetes or not.
* This separation is a common initial step for preparing data for modeling.

Split the Dataset into Training and Testing Sets:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

* The dataset is further divided into training and testing sets, with 80% of the data used for training the model and 20% for evaluating its performance.
* The `random\_state` parameter ensures reproducibility of the split.

Feature Engineering and Preprocessing:

scaler = StandardScaler()

X\_train[numerical\_features] = scaler.fit\_transform(X\_train[numerical\_features])

X\_test[numerical\_features] = scaler.transform(X\_test[numerical\_features])

encoder = OneHotEncoder(drop='first', sparse=False)

X\_train\_encoded = encoder.fit\_transform(X\_train[categorical\_features])

X\_test\_encoded = encoder.transform(X\_test[categorical\_features])

X\_train = np.concatenate([X\_train.drop(categorical\_features, axis=1).values, X\_train\_encoded], axis=1)

X\_test = np.concatenate([X\_test.drop(categorical\_features, axis=1).values, X\_test\_encoded], axis=1)

* In this step, feature preprocessing is performed.
* Numerical features are standardized using the StandardScaler to have a mean of 0 and a standard deviation of 1. This ensures that all numerical features are on a common scale.
* Categorical feature 'BMI\_Category' is one-hot encoded, converting it into a numerical format. This is important because machine learning models typically work with numerical data.

**Model Training and Hyperparameter Tuning:**

param\_grid = {

'n\_estimators': [50, 100, 150],

'max\_depth': [None, 10, 20, 30],

'min\_samples\_split': [2, 5, 10],

'min\_samples\_leaf': [1, 2, 4]

}

rf = RandomForestClassifier(random\_state=42)

grid\_search = GridSearchCV(estimator=rf, param\_grid=param\_grid, cv=5, scoring='accuracy')

grid\_search.fit(X\_train, y\_train)

best\_model = grid\_search.best\_estimator\_

* This step involves hyperparameter tuning for the Random Forest Classifier.
* A grid of hyperparameters is defined, and GridSearchCV is used to perform a search across this grid.
* Cross-validation with a 5-fold cross-validation strategy is applied to evaluate model performance for different hyperparameter combinations.
* The best model is selected based on the cross-validation results.

**Model Evaluation with Cross-Validation:**

cv\_scores = cross\_val\_score(best\_model, X\_train, y\_train, cv=5, scoring='accuracy')

print("Cross-Validation Scores:", cv\_scores)

print("Mean CV Accuracy:", np.mean(cv\_scores))

* The best model selected after hyperparameter tuning is evaluated using cross-validation.
* This provides a more robust assessment of the model's performance by testing it on different subsets of the training data.
* The cross-validation scores and the mean cross-validation accuracy are displayed, giving an indication of how well the model generalizes to new data.