**CODE FOR DISEASE PREDICTION**

**STEP1:**

#Importing the libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

# Ignore harmless warnings

import warnings

warnings.filterwarnings("ignore")

# Set to display all the columns in dataset

pd.set\_option("display.max\_columns", None)

# Import psql to run queries

import pandasql as psql

**STEP 2:**

#load the disease\_testing datset

disease=pd.read\_csv(r"C:\Users\harsh\Downloads\disease\_Testing.csv",header=0)

#copy file to backup file

disease\_BK=disease.copy()

#display the first 5 records

disease.head()

**STEP 3:**

disease.info()

**STEP 4:**

disease.shape

**STEP 5:**

#displaying the duplicate values in dataset

disease\_dup = disease[disease.duplicated(keep='last')]

disease\_dup

**STEP 6:**

disease.nunique()

**STEP 7:**

disease.isnull().sum()

**STEP 8:**

disease.columns

**STEP 9:**

disease['prognosis'].value\_counts()

**STEP 10:**

#use label encoder for target variables

from sklearn.preprocessing import LabelEncoder

LE=LabelEncoder()

disease['prognosis']=LE.fit\_transform(disease['prognosis'])

**STEP 11:**

disease.head()

**STEP 12:**

# Identify the independent and Target (dependent) variables

IndepVar = []

for col in disease.columns:

if col != 'prognosis':

IndepVar.append(col)

TargetVar = 'prognosis'

x = disease[IndepVar]

y = disease[TargetVar]

**STEP 13:**

# Split the data into train and test (random sampling)

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3, random\_state=42)

# Display the shape for train & test data

x\_train.shape, x\_test.shape, y\_train.shape, y\_test.shape

**STEP 14:**

x\_train.head()

**STEP 15:**

y\_train.head()

**STEP 16:**

from sklearn.preprocessing import MinMaxScaler

mmscaler = MinMaxScaler(feature\_range=(0, 1))

x\_train = mmscaler.fit\_transform(x\_train)

x\_train = pd.DataFrame(x\_train)

x\_test = mmscaler.fit\_transform(x\_test)

x\_test = pd.DataFrame(x\_test)

**STEP 18:**

#load the result dataset

medical\_results=pd.read\_csv(r"C:\Users\harsh\Downloads\knnresults.csv",header=0)

medical\_results.head()

**STEP 19:**

**pip install pandas numpy scikit-learn**

**pip install imbalanced-learn xgboost openpyxl**

**import pandas as pd**

**import numpy as np**

**from sklearn.model\_selection import train\_test\_split, GridSearchCV**

**from sklearn.preprocessing import StandardScaler, LabelEncoder**

**from sklearn.neighbors import KNeighborsClassifier**

**from sklearn.tree import DecisionTreeClassifier**

**from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, VotingClassifier**

**from sklearn.linear\_model import LogisticRegression**

**from sklearn.metrics import accuracy\_score, classification\_report**

**from imblearn.over\_sampling import SMOTE**

**import xgboost as xgb**

**# Load the dataset**

**file\_path = r"C:\Users\ratho\OneDrive\Desktop\Medical\_dataset(1).xlsx"**

**data = pd.read\_excel(file\_path, header=0)**

**# Preprocessing**

**le = LabelEncoder()**

**data['Gender'] = le.fit\_transform(data['Gender'])**

**data['Family Medical History'] = le.fit\_transform(data['Family Medical History'])**

**# Splitting features and target variable**

**X = data.drop('Family Medical History', axis=1)**

**y = data['Family Medical History']**

**# Splitting 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)**

**# Feature scaling**

**scaler = StandardScaler()**

**X\_train = scaler.fit\_transform(X\_train)**

**X\_test = scaler.transform(X\_test)**

**# Handle class imbalance using SMOTE**

**smote = SMOTE(random\_state=42)**

**X\_train\_res, y\_train\_res = smote.fit\_resample(X\_train, y\_train)**

**# Hyperparameter tuning for Random Forest, XGBoost, and Gradient Boosting**

**param\_grid\_rf = {**

**'n\_estimators': [100, 200, 300],**

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

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

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

**}**

**param\_grid\_xgb = {**

**'n\_estimators': [100, 200, 300],**

**'max\_depth': [3, 6, 9],**

**'learning\_rate': [0.01, 0.1, 0.2],**

**'subsample': [0.8, 1.0],**

**'colsample\_bytree': [0.8, 1.0]**

**}**

**param\_grid\_gb = {**

**'n\_estimators': [100, 200, 300],**

**'learning\_rate': [0.01, 0.1, 0.2],**

**'max\_depth': [3, 6, 9],**

**'subsample': [0.8, 1.0]**

**}**

**# Models initialization with refined hyperparameter tuning**

**models = {**

**'KNN': KNeighborsClassifier(n\_neighbors=5), # Adjust KNN parameters**

**'Decision Tree': DecisionTreeClassifier(random\_state=42),**

**'Random Forest': GridSearchCV(RandomForestClassifier(random\_state=42), param\_grid\_rf, cv=3, n\_jobs=-1, verbose=2),**

**'Logistic Regression': LogisticRegression(random\_state=42, max\_iter=1000),**

**'XGBoost': GridSearchCV(xgb.XGBClassifier(random\_state=42), param\_grid\_xgb, cv=3, n\_jobs=-1, verbose=2),**

**'Gradient Boosting': GridSearchCV(GradientBoostingClassifier(random\_state=42), param\_grid\_gb, cv=3, n\_jobs=-1, verbose=2)**

**}**

**# Function to evaluate models**

**def evaluate\_model(model, X\_train, y\_train, X\_test, y\_test):**

**model.fit(X\_train, y\_train)**

**y\_pred = model.predict(X\_test)**

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

**report = classification\_report(y\_test, y\_pred)**

**return accuracy, report**

**# Evaluating each model**

**results = {}**

**for model\_name, model in models.items():**

**print(f"Training {model\_name}...")**

**accuracy, report = evaluate\_model(model, X\_train\_res, y\_train\_res, X\_test, y\_test)**

**results[model\_name] = {'Accuracy': accuracy, 'Classification Report': report**

**# Displaying results**

**for model\_name, result in results.items():**

**print(f"Model: {model\_name}")**

**print(f"Accuracy: {result['Accuracy']:.2f}")**

**print("Classification Report:")**

**print(result['Classification Report'])**

**print("-" \* 50)**

**# Plotting bar graph for accuracies**

**model\_names = list(results.keys())**

**accuracies = [results[model]['Accuracy'] for model in model\_names]**

**plt.figure(figsize=(10, 6))**

**plt.bar(model\_names, accuracies, color='skyblue')**

**plt.xlabel('Models')**

**plt.ylabel('Accuracy')**

**plt.title('Accuracy Comparison of Models')**

**plt.ylim([0, 1])**

**plt.xticks(rotation=45)**

**plt.tight\_layout()**

**plt.show()**