Diabetes Risk Detection Machine Learning Project

Introduction

This report introduces the Diabetes Risk Detection project, a comprehensive approach to utilizing machine learning for identifying individuals at risk of diabetes by analyzing various health indicators. This report outlines the methodologies and strategies employed in leveraging a dataset that encompasses crucial parameters such as BMI, age, smoking status, and physical activity to construct and train a supervised model capable of forecasting diabetes likelihood. Additionally, the project aims to develop a system for predicting diabetes risk based on individual health indicators.

Key aspects of the project include:

1 Utilization of advanced data preprocessing and cleansing methodologies.

2 Conduct of thorough Exploratory Data Analysis (EDA) to uncover intricate interrelationships within the dataset.

3 Implementation of sophisticated feature selection and engineering techniques to prepare the data for modeling.

4 Assessment and comparison of diverse machine learning models, including Logistic Regression and Random Forest, to determine the most effective predictor of diabetes risk.

5 Integration of an interactive interface for user input and prediction generation.

Key Value and Potential Use Cases of This Project:

The Diabetes Risk Detection project holds significant value in healthcare analytics for predicting disease risk and facilitating early intervention strategies. By accurately identifying individuals predisposed to diabetes, this project offers potential use cases such as:

. Early intervention programs targeted towards high-risk individuals.

. Personalized healthcare plans tailored to mitigate diabetes risk factors.

. Resource allocation optimization within healthcare systems for proactive management of diabetes-related complications.

. Includes an interactive interface for user input and prediction generation.

Summary of the Different Phases of This Project and Milestones:

1 Data Preprocessing: Extraction of relevant features from the dataset and creation of a balanced binary dataset suitable for machine learning algorithms.

2 Model Development: Construction and training of a supervised machine learning model using the preprocessed dataset to predict diabetes likelihood.

3 Evaluation and Comparison: Assessment and juxtaposition of various machine learning models to identify the most effective predictor of diabetes risk.

4 Validation and Deployment: Validation of the selected model using independent datasets and deployment in real-world healthcare settings for practical application.

Milestones:

. Completion of data preprocessing phase and creation of balanced binary dataset.

. Development and training of machine learning models for diabetes risk prediction.

. Evaluation and comparison of models to select the most effective predictor.

. Validation of the selected model and deployment in real-world healthcare environments.

This structured approach ensures systematic progress towards achieving the project objectives while maintaining alignment with healthcare analytics goals.

Part 1

Data Preprocessing

Data Preparation and Initial Analysis:

This section aims at cleaning, processing, optimizing and exporting the datasets that we'll be using in the next part

1 Data Cleaning

2 Making features names more readable

3 Save the cleaned Datasets

4 Creating the binary Dataset

5 Exporting the Datasets

Entrée [1]:
```
#Imports necessary libraries for data analysis and visualization.

import numpy as np  # Importing the numpy library and aliasing it as 'np'
import pandas as pd  # Importing the pandas library and aliasing it as 'pd'
from scipy import stats  # Importing the stats module from the scipy library
import seaborn as sns  # Importing the seaborn library and aliasing it as 'sns'
from IPython.core.display import HTML  # Importing the HTML class from the IPython.core.display module
import matplotlib.pyplot as plt  # Importing the pyplot module from the matplotlib library and aliasing it as 'plt'
from scipy.stats import uniform  # Importing the uniform distribution function from the stats module of scipy
```

Entrée [2]:
```
#Reading the dataset

data = pd.read_csv(r'C:\Users\HELIOS-300\Downloads\Heart_Disease_Dataset\LLCP2021.csv')
pd.set_option('display.max_columns', 500)
data.head()
```

Out[2]:

	_STATE	FMONTH	IDATE	IMONTH	IDAY	IYEAR	DISPCODE	SEQNO	_PSU	CTELENM1	PVTRESD1	COLGHOUS	STATERE1	CELPHON1
0	1	1	1192021	1	19	2021	1100	2021000001	2021000001	1.0	1.0	NaN	1.0	2.0
1	1	1	1212021	1	21	2021	1100	2021000002	2021000002	1.0	1.0	NaN	1.0	2.0
2	1	1	1212021	1	21	2021	1100	2021000003	2021000003	1.0	1.0	NaN	1.0	2.0
3	1	1	1172021	1	17	2021	1100	2021000004	2021000004	1.0	1.0	NaN	1.0	2.0
4	1	1	1152021	1	15	2021	1100	2021000005	2021000005	1.0	1.0	NaN	1.0	2.0

Entrée [3]:
```
#viewing the dataset shape

data.shape
```

Out[3]: (438693, 303)

Entrée [4]:
```
#Select specific columns

data_selected = data[['DIABETE4', '_RFHYPE6', 'TOLDHI3', '_CHOLCH3', '_BMI5', 'SMOKE100', 'CVDSTRK3', '_MICHD', '_TOTINDA',
```

Entrée [5]:
```
#See how many rows and columns left

data_selected.head()
```

Out[5]:

	DIABETE4	_RFHYPE6	TOLDHI3	_CHOLCH3	_BMI5	SMOKE100	CVDSTRK3	_MICHD	_TOTINDA	_FRTLT1A	_VEGLT1A	_RFDRHV7	_HLTHPLN	GEN
0	3.0	1	1.0	1	1454.0	1.0	2.0	2.0	2	1	1	1	1	
1	1.0	2	1.0	1	NaN	2.0	2.0	1.0	1	1	1	1	1	
2	1.0	2	2.0	1	2829.0	2.0	2.0	1.0	2	1	2	1	1	
3	1.0	2	1.0	1	3347.0	2.0	2.0	2.0	1	1	1	1	1	
4	1.0	1	1.0	1	2873.0	2.0	1.0	1.0	1	1	1	1	1	

Data Cleaning:

Drop missing values

Entrée [6]:
```
data_selected = data_selected.dropna()
data_selected.shape
```

Out[6]: (330361, 21)

Entrée [7]:
```
#Checking the data tyeps

data.dtypes
```

Out[7]:
```
_STATE      int64
FMONTH      int64
IDATE       int64
IMONTH      int64
IDAY        int64
            ...
_VEGLT1A    int64
_FRT16A     int64
_VEG23A     int64
_FRUITE1    int64
_VEGETE1    int64
Length: 303, dtype: object
```

Modify and clean the values to be more suitable to ML algorithms

Entrée [8]:
```
# DIABETE4 = Diabetes Awareness
# going to make this ordinal. 0 is for no diabetes or only during pregnancy, 1 is for yes diabetes
# Remove all 4 (No, pre-diabetes)
# Remove all 7 (dont knows)
# Remove all 9 (refused)
data_selected['DIABETE4'] = data_selected['DIABETE4'].replace({2:0, 3:0})
data_selected = data_selected[data_selected.DIABETE4 != 4]
data_selected = data_selected[data_selected.DIABETE4 != 7]
data_selected = data_selected[data_selected.DIABETE4 != 9]
data_selected.DIABETE4.unique()
```

Out[8]:
```
array([0., 1.])
```

Entrée [9]:
```
#1 _RFHYPE6 = High Blood Pressure Awareness
#Change 1 to 0 so it represetnts No high blood pressure and 2 to 1 so it represents high blood pressure
data_selected['_RFHYPE6'] = data_selected['_RFHYPE6'].replace({1:0, 2:1})
data_selected = data_selected[data_selected._RFHYPE6 != 9]
data_selected._RFHYPE6.unique()
```

Out[9]:
```
array([0, 1], dtype=int64)
```

Entrée [10]:
```
# TOLDHI3 = Cholesterol Awareness
# Change 2 to 0 because it is No
# Remove all 7 (dont knows)
# Remove all 9 (refused)
data_selected['TOLDHI3'] = data_selected['TOLDHI3'].replace({2:0})
data_selected = data_selected[data_selected.TOLDHI3 != 7]
data_selected = data_selected[data_selected.TOLDHI3 != 9]
data_selected.TOLDHI3.unique()
```

Out[10]:
```
array([1., 0.])
```

Entrée [11]:
```
# _CHOLCH3 = Cholesterol check within past five years
# Change 3 to 0 and 2 to 0 for Not checked cholesterol in past 5 years
# Remove 9 (don't know/refused)
data_selected['_CHOLCH3'] = data_selected['_CHOLCH3'].replace({3:0,2:0})
data_selected = data_selected[data_selected._CHOLCH3 != 9]
data_selected._CHOLCH3.unique()
```

Out[11]:
```
array([1, 0], dtype=int64)
```

Entrée [12]:
```
# SMOKE100 = Smoked at Least 100 Cigarettes
# Change 2 to 0 because it is No
# Remove all 7 (dont knows)
# Remove all 9 (refused)
data_selected['SMOKE100'] = data_selected['SMOKE100'].replace({2:0})
data_selected = data_selected[data_selected.SMOKE100 != 7]
data_selected = data_selected[data_selected.SMOKE100 != 9]
data_selected.SMOKE100.unique()
```

Out[12]:
```
array([1., 0.])
```

Entrée [13]:
```
# CVDSTRK3 = Chronic Health Conditions
# Change 2 to 0 because it is No
# Remove all 7 (dont knows)
# Remove all 9 (refused)
data_selected['CVDSTRK3'] = data_selected['CVDSTRK3'].replace({2:0})
data_selected = data_selected[data_selected.CVDSTRK3 != 7]
data_selected = data_selected[data_selected.CVDSTRK3 != 9]
data_selected.CVDSTRK3.unique()
```

Out[13]:
```
array([0., 1.])
```

Entrée [14]:
```
# _MICHD = Ever had CHD or MI
# Coronary Heart Disease (CHD) and Myocardial Infarction (MI)
# Change 2 to 0 because this means they did not have MI or CHD
data_selected['_MICHD'] = data_selected['_MICHD'].replace({2: 0})
data_selected._MICHD.unique()
```

Out[14]:
```
array([0., 1.])
```

Entrée [15]:
```
# _TOTINDA = Leisure Time Physical Activity
# Change 2 to 0 for no physical activity
# Remove all 9 (don't know/refused)
data_selected['_TOTINDA'] = data_selected['_TOTINDA'].replace({2:0})
data_selected = data_selected[data_selected._TOTINDA != 9]
data_selected._TOTINDA.unique()
```

Out[15]:
```
array([0, 1], dtype=int64)
```

Entrée [16]:
```
# _FRTLT1A = Consume Fruit 1 or more per day
# Change 2 to 0. this means no fruit consumed per day. 1 will mean consumed 1 or more pieces of fruit per day
# Remove all 9 (don't know/refused)
data_selected['_FRTLT1A'] = data_selected['_FRTLT1A'].replace({2:0})
data_selected = data_selected[data_selected._FRTLT1A != 9]
data_selected._FRTLT1A.unique()
```

Out[16]:
```
array([1, 0], dtype=int64)
```

Entrée [17]:
```
# _VEGLT1A = Consume Vegetables 1 or more per day
# Change 2 to 0. this means no vegetables consumed per day. 1 will mean consumed 1 or more pieces of vegetable per day
# Remove all 9 (don't know/refused)
data_selected['_VEGLT1A'] = data_selected['_VEGLT1A'].replace({2:0})
data_selected = data_selected[data_selected._VEGLT1A != 9]
data_selected._VEGLT1A.unique()
```

Out[17]:
```
array([1, 0], dtype=int64)
```

Entrée [18]:
```
# _RFDRHV7 = Heavy Alcohol Consumption
# Change 1 to 0 (1 was no for heavy drinking). change all 2 to 1 (2 was yes for heavy drinking)
# Remove all 9 (don't know/refused)
data_selected['_RFDRHV7'] = data_selected['_RFDRHV7'].replace({1:0, 2:1})
data_selected = data_selected[data_selected._RFDRHV7 != 9]
data_selected._RFDRHV7.unique()
```

Out[18]:
```
array([0, 1], dtype=int64)
```

Entrée [19]:
```
# _HLTHPLN = Have any health insurance
# Change 2 to 0 for no health insurance
# Remove all 9 (don't know/refused)
data_selected['_HLTHPLN'] = data_selected['_HLTHPLN'].replace({2:0})
data_selected = data_selected[data_selected._HLTHPLN != 7]
data_selected = data_selected[data_selected._HLTHPLN != 9]
data_selected._HLTHPLN.unique()
```

Out[19]:
```
array([1, 0], dtype=int64)
```

Entrée [20]:
```
# GENHLTH = General Health
# 1 is Excellent -> 5 is Poor
# Remove all 7 (dont knows)
# Remove all 9 (refused)
data_selected = data_selected[data_selected.GENHLTH != 7]
data_selected = data_selected[data_selected.GENHLTH != 9]
data_selected.GENHLTH.unique()
```

Out[20]:
```
array([5., 2., 3., 4., 1.])
```

Entrée [21]:
```
# MENTHLTH = Number of Days Mental Health is Not Good
# It is already in days, scale will be between 0-30
# Change 88 to 0 because it means none (no bad mental health days)
# Remove all 77 (dont knows)
# Remove all 99 (refused)
data_selected['MENTHLTH'] = data_selected['MENTHLTH'].replace({88:0})
data_selected = data_selected[data_selected.MENTHLTH != 77]
data_selected = data_selected[data_selected.MENTHLTH != 99]
data_selected.MENTHLTH.unique()
```

Out[21]:
```
array([10.,  0.,  5., 25.,  2.,  7., 30., 14., 20.,  8.,  1.,  3., 15.,
        4., 28., 24., 21., 12.,  6., 22., 27., 18., 13., 17., 16.,  9.,
       19., 29., 23., 11., 26.])
```

Entrée [22]:
```
# PHYSHLTH = Number of Days Physical Health is Not Good
# It is already in days, scale will be between 0-30
# Change 88 to 0 because it means none (no bad physical health days)
# Remove all 77 (dont knows)
# Remove all 99 (refused)
data_selected['PHYSHLTH'] = data_selected['PHYSHLTH'].replace({88:0})
data_selected = data_selected[data_selected.PHYSHLTH != 77]
data_selected = data_selected[data_selected.PHYSHLTH != 99]
data_selected.PHYSHLTH.unique()
```

Out[22]:
```
array([20.,  0., 30., 25.,  1.,  4., 10.,  2.,  3., 15.,  8., 14.,  5.,
        7.,  6., 24., 29., 18.,  9., 16., 17., 26., 28., 12., 27., 13.,
       21., 11., 19., 22., 23.])
```

Entrée [23]:
```
# DIFFWALK = Difficulty Walking or Climbing Stairs
# Change 2 to 0 for no.
# Remove all 7 (dont knows)
# Remove all 9 (refused)
data_selected['DIFFWALK'] = data_selected['DIFFWALK'].replace({2:0})
data_selected = data_selected[data_selected.DIFFWALK != 7]
data_selected = data_selected[data_selected.DIFFWALK != 9]
data_selected.DIFFWALK.unique()
```

Out[23]:
```
array([0., 1.])
```

Entrée [24]:
```
# _SEX = Respondent Sex
# Men are at higher risk for heart disease
# Change 2 to 0 (female as 0)
data_selected['_SEX'] = data_selected['_SEX'].replace({2:0})
data_selected._SEX.unique()
```

Out[24]: `array([0, 1], dtype=int64)`

Entrée [25]:
```
# _AGEG5YR = Reported age in five-year age categories
# 5 year increments. It is already ordinal. 1 is 18-24 all the way up to 13 which is 80 and older
# Remove all 14 (don't know or missing)
data_selected = data_selected[data_selected._AGEG5YR != 14]
data_selected._AGEG5YR.unique()
```

Out[25]: `array([11,  9, 12, 13, 10,  7,  6,  8,  4,  3,  5,  2,  1], dtype=int64)`

Entrée [26]:
```
# EDUCA = Education Level
# This is already an ordinal variable with 1 being never attended school or kindergarten only up to 6 being college 4 years o
# Scale here is 1-6
# Remove all 9 (refused)
data_selected = data_selected[data_selected.EDUCA != 9]
data_selected.EDUCA.unique()
```

Out[26]: `array([4., 3., 5., 6., 2., 1.])`

Entrée [27]:
```
# _INCOMG1 =  Computed income categories
# Remove all 9 (refused)
data_selected = data_selected[data_selected._INCOMG1 != 9]
data_selected._INCOMG1.unique()
```

Out[27]: `array([3, 2, 5, 4, 1, 6, 7], dtype=int64)`

Entrée [28]:
```
data_selected.shape
```

Out[28]: `(231037, 21)`

Entrée [29]:
```
data_selected.head()
```

Out[29]:

	DIABETE4	_RFHYPE6	TOLDHI3	_CHOLCH3	_BMI5	SMOKE100	CVDSTRK3	_MICHD	_TOTINDA	_FRTLT1A	_VEGLT1A	_RFDRHV7	_HLTHPLN	GEN
0	0.0	0	1.0	1	1454.0	1.0	0.0	0.0	0	1	1	0	1	
2	1.0	1	0.0	1	2829.0	0.0	0.0	1.0	0	1	0	0	1	
3	1.0	1	1.0	1	3347.0	0.0	0.0	0.0	1	1	1	0	1	
4	1.0	0	1.0	1	2873.0	0.0	1.0	1.0	1	1	1	0	1	
5	0.0	0	0.0	1	2437.0	1.0	0.0	0.0	0	0	0	0	1	

Entrée [30]:
```
#Check Class Sizes of the heart disease column
data_selected.groupby(['DIABETE4']).size()
```

Out[30]:
```
DIABETE4
0.0    197428
1.0     33609
dtype: int64
```

Make feature names more readable

Entrée [31]:
```
#Rename the columns to make them more readable
data_cleaned = data_selected.rename(columns = {'DIABETE4':'Has_Diabetes', '_RFHYPE6':'HighBP', 'TOLDHI3':'HighChol', '_CHOLCH
```

Entrée [32]:
```
data_cleaned.head()
```

Out[32]:

	Has_Diabetes	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttack	PhysActivity	Fruits	Vegetables	HvyAlcoholConsump	Any
0	0.0	0	1.0	1	1454.0	1.0	0.0	0.0	0	1	1	0	
2	1.0	1	0.0	1	2829.0	0.0	0.0	1.0	0	1	0	0	
3	1.0	1	1.0	1	3347.0	0.0	0.0	0.0	1	1	1	0	
4	1.0	0	1.0	1	2873.0	0.0	1.0	1.0	1	1	1	0	
5	0.0	0	0.0	1	2437.0	1.0	0.0	0.0	0	0	0	0	

Entrée [33]:
```
data_cleaned.shape
```

Out[33]: `(231037, 21)`

Entrée [34]:
```
#Check how many respondents have no diabetes, prediabetes or diabetes. Note the class imbalance!
data_cleaned.groupby(['Has_Diabetes']).size()
```

Out[34]:
```
Has_Diabetes
0.0    197428
1.0     33609
dtype: int64
```

Save the cleaned dataset to a csv file

Entrée [35]:
```
data_cleaned.to_csv('diabetes_health_indicators_s1.csv', sep=",", index=False)
```

Creating a Binary Dataset for diabetes vs. no diabetes

Entrée [36]:
```
#Copy old table to a new one.
diabetes_binary = data_cleaned

#Change the column name to Diabetes_binary
diabetes_binary = diabetes_binary.rename(columns = {'Has_Diabetes': 'Diabetes_binary'})
diabetes_binary.Diabetes_binary.unique()
```

Out[36]:
```
array([0., 1.])
```

Entrée [37]:
```
#Show class sizes
diabetes_binary.groupby(['Diabetes_binary']).size()
```

Out[37]:
```
Diabetes_binary
0.0    197428
1.0     33609
dtype: int64
```

Entrée [38]:
```
#Separate the 0(No Diabetes) and 1&2(Pre-diabetes and Diabetes)
#Get the 1s
is1 = diabetes_binary['Diabetes_binary'] == 1
diabetes_binary_1 = diabetes_binary[is1]

#Get the 0s
is0 = diabetes_binary['Diabetes_binary'] == 0
diabetes_binary_0 = diabetes_binary[is0]

#Select the 33609 random cases from the 0 (non-diabetes group). we already have 33609 cases from the diabetes risk group
diabetes_binary_0_rand1 = diabetes_binary_0.take(np.random.permutation(len(diabetes_binary_0))[:33609])

#Append the 33609 1s to the 33609 randomly selected 0s
diabetes_5050 = diabetes_binary_0_rand1._append(diabetes_binary_1, ignore_index = True)
```

Entrée [39]:
```
diabetes_5050.head()
```

Out[39]:

	Diabetes_binary	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttack	PhysActivity	Fruits	Vegetables	HvyAlcoholConsump
0	0.0	0	1.0	1	2260.0	0.0	0.0	0.0	1	0	1	0
1	0.0	1	1.0	1	2912.0	0.0	0.0	0.0	1	0	1	0
2	0.0	0	0.0	1	2798.0	0.0	0.0	0.0	1	1	1	0
3	0.0	0	0.0	1	2609.0	0.0	0.0	0.0	1	1	1	1
4	0.0	1	0.0	1	2798.0	0.0	0.0	0.0	1	1	1	0

Entrée [40]:
```
diabetes_5050.groupby(['Diabetes_binary']).size()
```

Out[40]:
```
Diabetes_binary
0.0    33609
1.0    33609
dtype: int64
```

Save binary dataset and 50-50 binary balanced dataset to csv file

Entrée [41]:
```
print(f'diabetes_5050={diabetes_5050.shape}',f'diabetes_binary={diabetes_binary.shape}')
```

```
diabetes_5050=(67218, 21) diabetes_binary=(231037, 21)
```

Entrée [42]:
```
diabetes_5050.to_csv('diabetes_binary_5050split_health_indicators_s1.csv', sep=",", index=False)
```

Entrée [43]:
```
diabetes_binary.to_csv('diabetes_binary_health_indicators_s1.csv', sep=",", index=False)
```

Part 2

Machine Learning Models

This phase involves evaluating different machine learning algorithms for predicting diabetes risk using preprocessed datasets. Key steps include:

Key components of this phase include:

- Data Preparation and Initial Analysis.
- Data Type Optimizatioin Analysis and Conversion for Efficiency
- Logistic Regression Model
- Random Forest Model
- Corss Validation of Models

Data Preparation and Initial Analysis

Entrée [44]:
```
from IPython.display import display
from matplotlib import pyplot
from pandas.plotting import scatter_matrix
from sklearn import linear_model, metrics, model_selection
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    ConfusionMatrixDisplay,
    accuracy_score,
    classification_report,
    confusion_matrix,
)
from sklearn.model_selection import KFold, cross_val_score, train_test_split
from sklearn.preprocessing import MinMaxScaler

# file names and urls
filepath_2015 = r"C:\Users\HELIOS-300\Downloads\archive_(2)\diabetes_binary_5050split_health_indicators_BRFSS2015.csv"
filepath_2021 = r"C:\Users\HELIOS-300\Downloads\archive_(2)\diabetes_binary_5050split_health_indicators_BRFSS2021.csv"
df1 = pd.read_csv(filepath_2015)
df2 = pd.read_csv(filepath_2021)

# Combine the two DataFrames
combined_df = pd.concat([df1, df2], axis=0).reset_index(drop=True)

# Rename the 'Diabetes_binary' column to 'Diabetes'
combined_df.rename(columns={"Diabetes_binary": "Diabetes"}, inplace=True)

# Display the first few rows of the combined dataframe
# and its shape to verify the combination
combined_df_info = combined_df.head(), combined_df.shape

combined_df_info  # output
```

Out[44]:
```
(   Diabetes  HighBP  HighChol  CholCheck   BMI  Smoker  Stroke  \
 0       0.0     1.0       0.0        1.0  26.0     0.0     0.0   
 1       0.0     1.0       1.0        1.0  26.0     1.0     1.0   
 2       0.0     0.0       0.0        1.0  26.0     0.0     0.0   
 3       0.0     1.0       1.0        1.0  28.0     1.0     0.0   
 4       0.0     0.0       0.0        1.0  29.0     1.0     0.0   

    HeartDiseaseorAttack  PhysActivity  Fruits  Veggies  HvyAlcoholConsump  \
 0                   0.0           1.0     0.0      1.0                0.0   
 1                   0.0           0.0     1.0      0.0                0.0   
 2                   0.0           1.0     1.0      1.0                0.0   
 3                   0.0           1.0     1.0      1.0                0.0   
 4                   0.0           1.0     1.0      1.0                0.0   

    AnyHealthcare  NoDocbcCost  GenHlth  MentHlth  PhysHlth  DiffWalk  Sex  \
 0            1.0          0.0      3.0       5.0      30.0       0.0  1.0   
 1            1.0          0.0      3.0       0.0       0.0       0.0  1.0   
 2            1.0          0.0      1.0       0.0      10.0       0.0  1.0   
 3            1.0          0.0      3.0       0.0       3.0       0.0  1.0   
 4            1.0          0.0      2.0       0.0       0.0       0.0  0.0   

     Age  Education  Income  
 0   4.0        6.0     8.0  
 1  12.0        6.0     8.0  
 2  13.0        6.0     8.0  
 3  11.0        6.0     8.0  
 4   8.0        5.0     8.0  ,
 (137828, 22))
```

Removing Features

In this step, we eliminate features considered irrelevant for modeling by specifying a list of columns to be removed and dropping them from the combined DataFrame.

Additionally, we adjust the values of the "General Health" feature to enhance clarity for visualization. This entails reversing the values such that a value of

Entrée [45]:
```
# Remove irrelevant features from the combined dataset
columns_to_remove = ["CholCheck", "AnyHealthcare", "NoDocbcCost", "Education", "Income"]
reduced_df = combined_df.drop(columns=columns_to_remove)

# Reverse the values of 'GenHlth'
reduced_df["GenHlth"] = 6 - reduced_df["GenHlth"]

# Display the first few rows of the reduced dataframe to verify the removal
reduced_df.head()
```

Out[45]:

	Diabetes	HighBP	HighChol	BMI	Smoker	Stroke	HeartDiseaseorAttack	PhysActivity	Fruits	Veggies	HvyAlcoholConsump	GenHlth	MentHlth	PhysI
0	0.0	1.0	0.0	26.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	3.0	5.0	:
1	0.0	1.0	1.0	26.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	3.0	0.0	
2	0.0	0.0	0.0	26.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	5.0	0.0	·
3	0.0	1.0	1.0	28.0	1.0	0.0	0.0	1.0	1.0	1.0	0.0	3.0	0.0	
4	0.0	0.0	0.0	29.0	1.0	0.0	0.0	1.0	1.0	1.0	0.0	4.0	0.0	

Check for Missing Values

Check for Missing Values. Here we check for missing values in the combined dataset. Note there are NO missing values found.

Entrée [46]:
```
# Check for missing values in the reduced dataset
missing_values = reduced_df.isnull().sum()

missing_values  ## no missing values found
```

Out[46]:
```
Diabetes                0
HighBP                  0
HighChol                0
BMI                     0
Smoker                  0
Stroke                  0
HeartDiseaseorAttack    0
PhysActivity            0
Fruits                  0
Veggies                 0
HvyAlcoholConsump       0
GenHlth                 0
MentHlth                0
PhysHlth                0
DiffWalk                0
Sex                     0
Age                     0
dtype: int64
```

Data Type Optimization Analysis

Here we conduct an initial analysis to enhance data storage and processing efficiency by assessing the range of values for selected features within the reduced dataset. We observe that all numerical values fall within the range of an 8-bit integer (without decimal values in this dataset).

Furthermore, we compute the memory usage before adjusting the data types, establishing a baseline for memory consumption.

Entrée [47]:
```
# Check range of values of specified features to determine suitable data types
features_to_optimize = ["BMI", "GenHlth", "MentHlth", "PhysHlth", "Age"]
data_types_optimization = (
    reduced_df[features_to_optimize].describe().loc[["min", "max"]]
)

# Memory used before reducing data types
memory_before = reduced_df.memory_usage(index=True).sum()

data_types_optimization
```

Out[47]:

	BMI	GenHlth	MentHlth	PhysHlth	Age
min	12.0	1.0	0.0	0.0	1.0
max	99.0	5.0	30.0	30.0	13.0

Data Type Conversion for Efficiency

We improve the memory efficiency of the dataset by converting specified binary columns to boolean data types. To illustrate the effectiveness of this optimization in reducing memory consumption, we display the memory usage before and after the operation.

Entrée [48]:
```
# scale data types down to reduce memory footprint
reduced_df["BMI"] = reduced_df["BMI"].astype("float32")
reduced_df["GenHlth"] = reduced_df["GenHlth"].astype("int8")
reduced_df["MentHlth"] = reduced_df["MentHlth"].astype("int8")
reduced_df["PhysHlth"] = reduced_df["PhysHlth"].astype("int8")
reduced_df["Age"] = reduced_df["Age"].astype("int8")

# convert 1/0 binary columns to boolean values
binary_columns = [
    "Diabetes",
    "HighBP",
    "HighChol",
    "Smoker",
    "Stroke",
    "HeartDiseaseorAttack",
    "PhysActivity",
    "Fruits",
    "Veggies",
    "HvyAlcoholConsump",
    "DiffWalk",
    "Sex",
]
for column in binary_columns:
    reduced_df[column] = reduced_df[column].astype("bool")

# memory size after data type reduction
memory_after = reduced_df.memory_usage(index=True).sum()

print("Dataframe memory used before:", memory_before)
print("Dataframe memory used after:  ", memory_after)
```

```
Dataframe memory used before: 18744736
Dataframe memory used after:   2756688
```

Logistic Regression Model for Diabetes Prediction

Here, we prepare the data for machine learning, focusing on using a logistic regression model to predict diabetes. Initially, numerical columns are identified and scaled using MinMaxScaler to ensure all features contribute equally to the model without bias from varying scales. Subsequently, a logistic regression model is initialized with specific parameters. The dataset is split into features (X_log) and the target variable (y_log), followed by further division into training and test sets to evaluate the model's performance on unseen data.

Upon training the logistic regression model, predictions are generated on the test set. The model's effectiveness is then assessed using accuracy, confusion matrix, and classification report metrics, providing a comprehensive overview of its predictive capabilities in distinguishing between diabetic and non-diabetic individuals.

Entrée [49]:
```
# copy Dataframe for Logistic model
log_df = reduced_df.copy(deep=True)

# Selecting numerical columns (excluding binary/boolean columns)
numerical_columns = ["BMI", "GenHlth", "MentHlth", "PhysHlth", "Age"]

# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Fit and transform the numerical features
log_df[numerical_columns] = scaler.fit_transform(log_df[numerical_columns])

mylog_model = linear_model.LogisticRegression(solver="saga", max_iter=1000)

# 'X' is the feature set and 'y' is the target variable
X_log = log_df.drop("Diabetes", axis=1)
y_log = log_df["Diabetes"].astype("bool")  # Ensuring the target is boolean

# Splitting the dataset into the Training set and Test set
X_log_train, X_log_test, y_log_train, y_log_test = model_selection.train_test_split(
    X_log, y_log, test_size=0.25, random_state=42
)

# Train the model and output prediction of test data
mylog_model.fit(X_log_train, y_log_train)
y_pred_log = mylog_model.predict(X_log_test)

# Evaluate the model
accuracy_log = accuracy_score(y_log_test, y_pred_log)
conf_matrix_log = confusion_matrix(y_log_test, y_pred_log)
class_report_log = classification_report(y_log_test, y_pred_log)

print("\nLogistic Regression (single) prediction results:", "\n")
print(f"Accuracy: {round(accuracy_log*100,2)} %", "\n")
print("Confusion Matrix:")
print(conf_matrix_log, "\n")
print("Classification Report:")
print(class_report_log)
```

```

Logistic Regression (single) prediction results:

Accuracy: 74.58 %

Confusion Matrix:
[[12556  4657]
 [ 4101 13143]]

Classification Report:
              precision    recall  f1-score   support

       False       0.75      0.73      0.74     17213
        True       0.74      0.76      0.75     17244

    accuracy                           0.75     34457
   macro avg       0.75      0.75      0.75     34457
weighted avg       0.75      0.75      0.75     34457
```

Exporting the Model

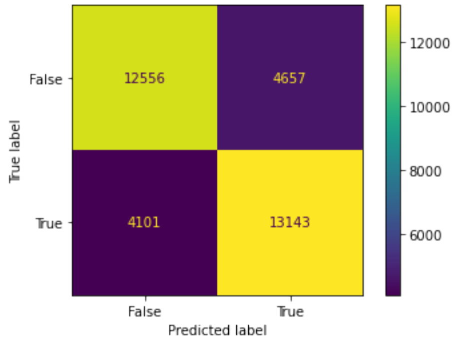
To save the trained model, we can use the joblib library. Here we show how to save and load the model.

Entrée [50]:
```
import joblib

# Save the model to a file
joblib.dump(mylog_model, 'diabetes-classifier-log_model.pkl')

# Later, load the model from the file with the following:
# mylog_model = joblib.load('my_model_filename.pkl')
```

Out[50]:
```
['diabetes-classifier-log_model.pkl']
```

Confusion Matrix Logistic Regression Model

Entrée [51]:
```
# Plot confusion matrix
graph_confusion_matrix = ConfusionMatrixDisplay.from_predictions(y_log_test, y_pred_log)
```



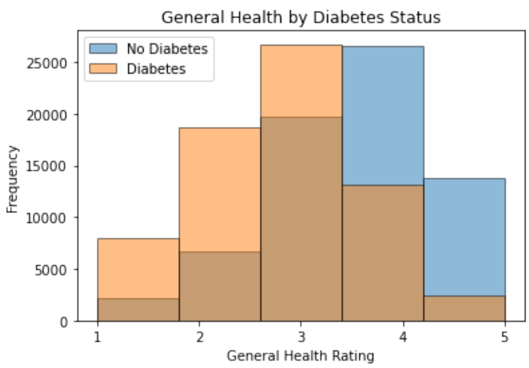
Data Exploration

In this section, we conduct data exploration through visualizations like histograms and scatterplots.

Histogram Figure 1.a:

We generate a histogram comparing the distribution of general health ratings (GenHlth) between individuals with and without diabetes (Diabetes status) in the dataset. For each group, we display the frequency of respondents across five health rating categories, ranging from poor to excellent health. These ratings have been reversed so that higher numbers indicate better health. The histogram for individuals without diabetes is displayed in blue, while the histogram for those with diabetes is shown in orange. This visualization facilitates a visual comparison of general health perceptions between the two groups, aiding in understanding whether there's a noticeable difference in self-reported general health status based on diabetes condition.

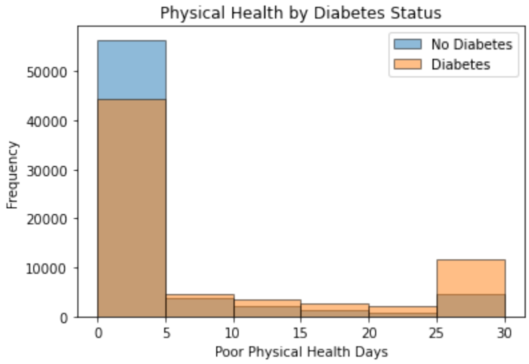
Entrée [52]:
```
# Filter the dataset by Diabetes status
gen_health_no_diabetes = reduced_df[reduced_df["Diabetes"] == False]["GenHlth"]
gen_health_with_diabetes = reduced_df[reduced_df["Diabetes"] == True]["GenHlth"]

# Plot histograms
plt.hist(
    gen_health_no_diabetes, bins=5, alpha=0.5, label="No Diabetes", edgecolor="black"
)
plt.hist(
    gen_health_with_diabetes, bins=5, alpha=0.5, label="Diabetes", edgecolor="black"
)

# Add legend
plt.legend()

# Add titles and labels as needed
plt.title("General Health by Diabetes Status")
plt.xlabel("General Health Rating")
plt.ylabel("Frequency")

# Set x-axis to display integer values from 1 to 5
plt.xticks(range(1, 6))

# Show the plot
plt.show()
```



Histogram Figure 1.b

Analysis of physical health comparing diabetics to non-diabetics.

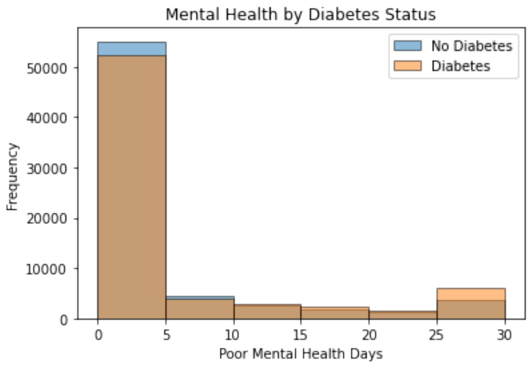
Entrée [53]:
```
# Filter the dataset by Diabetes status
phys_health_no_diabetes = reduced_df[reduced_df["Diabetes"] == False]["PhysHlth"]
phys_health_with_diabetes = reduced_df[reduced_df["Diabetes"] == True]["PhysHlth"]

# Plot histograms
plt.hist(
    phys_health_no_diabetes, alpha=0.5, label="No Diabetes", bins=6, edgecolor="black"
)
plt.hist(
    phys_health_with_diabetes, alpha=0.5, label="Diabetes", bins=6, edgecolor="black"
)

# Add legend
plt.legend()

# Add titles and labels as needed
plt.title("Physical Health by Diabetes Status")
plt.xlabel("Poor Physical Health Days")
plt.ylabel("Frequency")

# Show the plot
plt.show()
```



Histogram Figure 1.c

Analysis of mental health comparing diabetics to non-diabetics.

Entrée [54]:
```
# Filter the dataset by Diabetes status
phys_health_no_diabetes = reduced_df[reduced_df["Diabetes"] == False]["MentHlth"]
phys_health_with_diabetes = reduced_df[reduced_df["Diabetes"] == True]["MentHlth"]

# Plot histograms
plt.hist(
    phys_health_no_diabetes, alpha=0.5, label="No Diabetes", bins=6, edgecolor="black"
)
plt.hist(
    phys_health_with_diabetes, alpha=0.5, label="Diabetes", bins=6, edgecolor="black"
)

# Add legend
plt.legend()

# Add titles and labels as needed
plt.title("Mental Health by Diabetes Status")
plt.xlabel("Poor Mental Health Days")
plt.ylabel("Frequency")

# Show the plot
plt.show()
```

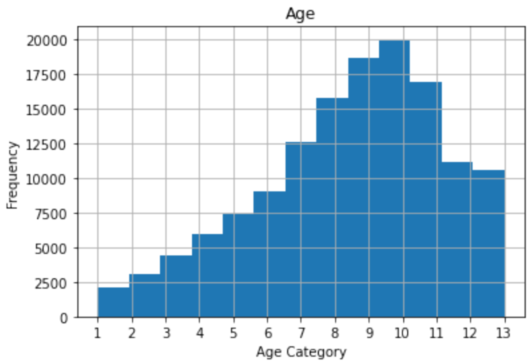


Histogram Figure 1.d

Age Distribution Analysis

This visualization presents a distribution analysis of the Age feature. Notably, the data was collected using a 13-level age category where 1 corresponds to the age group 18-24, 9 represents the age group 60-64, and 13 denotes individuals aged 80 or older.

Entrée [55]:
```
# Plot Age histogram
graph_histograms = reduced_df.hist(column="Age", grid=True, bins=13)

# Calculate the tick positions for 13 bins
tick_positions = range(1, 14)

# Set the x-axis ticks
plt.xticks(tick_positions)

# Optionally, set x-axis and y-axis labels
plt.xlabel("Age Category")
plt.ylabel("Frequency")

# Show the plot
plt.show()
```

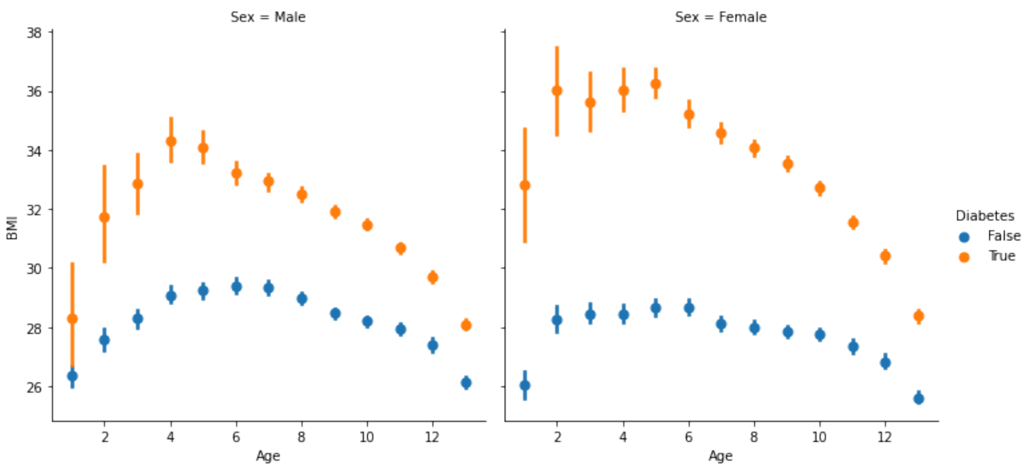


Scatterplot figure 2.a

Here we see that diabetics have a higher BMI on average.

Entrée [56]:
```
# Copy the DataFrame to avoid modifying the original data
plot_df = reduced_df.copy()

# Map True/False to Male/Female
plot_df["Sex"] = plot_df["Sex"].map({True: "Male", False: "Female"})

# Age/BMI Scatterplot
sns.lmplot(
    data=plot_df,
    x="Age",
    y="BMI",
    col="Sex",
    hue="Diabetes",
    x_bins=1000,
    fit_reg=False,
)
```

Out[56]: <seaborn.axisgrid.FacetGrid at 0x24d03bffa00>



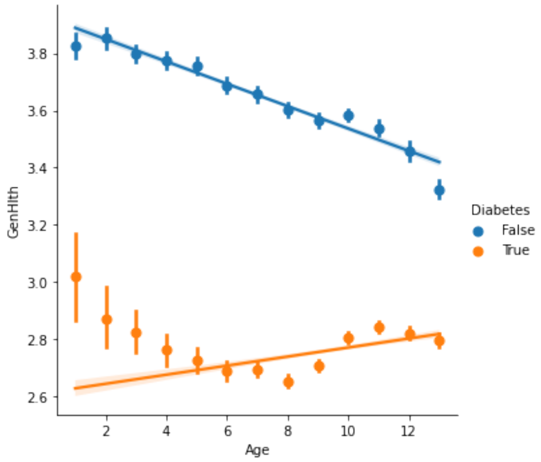
Scatterplot figure 2.b

General Health over Age:

The general health indicator is rated on a scale of 1 to 5, with a score of 5 representing excellent health. In this analysis, we observe that individuals with diabetes tend to report poorer general health compared to non-diabetics. Additionally, there is a noticeable trend among healthy individuals where general

Entrée [57]:
```python
# Age/General-Health Scatterplot
sns.lmplot(data=reduced_df, x="Age", y="GenHlth", hue="Diabetes", x_bins=1000)
```

Out[57]: <seaborn.axisgrid.FacetGrid at 0x24d03bad400>



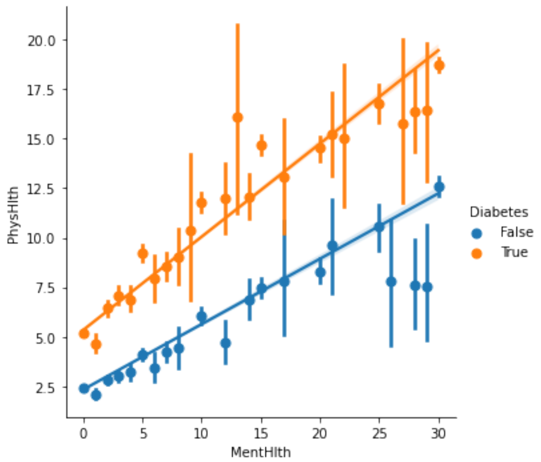
Scatterplot figure 2.c

Comparison of Physical Health to Mental Health:

In this analysis, we observe a trend indicating that individuals who experience more days per month of poor mental health tend to also experience more days of poor physical health, on average. Notably, individuals with diabetes exhibit a similar distribution of poor mental health days compared to non-diabetics, but report significantly more days per month of poor physical health. This suggests a potential association between mental and physical health outcomes, with diabetes potentially exacerbating the impact on physical health.

Entrée [58]:
```python
# Health Scatterplot
sns.lmplot(data=reduced_df, x="MentHlth", y="PhysHlth", hue="Diabetes", x_bins=1000)
```

Out[58]: <seaborn.axisgrid.FacetGrid at 0x24d03b70820>



Random Forest Model for Comparison

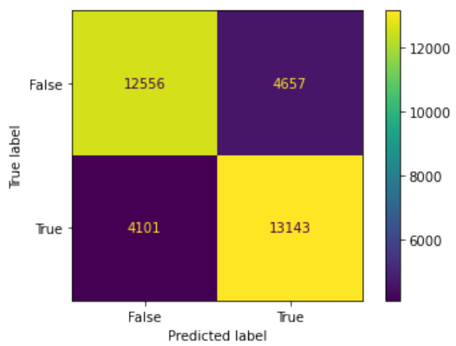
Here, we establish a second model using the Random Forest algorithm for comparison with our Logistic Regression model. The process follows similar steps as before, with the exception that scaling of numerical data is not required for Random Forest algorithm.

Entrée [59]:
```
# 'X' is the set of features and 'y' is the target variable
X_rf = reduced_df.drop("Diabetes", axis=1)
y_rf = reduced_df["Diabetes"].astype("bool")  # Ensuring the target is boolean

# Splitting the dataset into the Training set and Test set
X_rf_train, X_rf_test, y_rf_train, y_rf_test = train_test_split(
    X_rf, y_rf, test_size=0.25, random_state=42
)

# Creating a Random Forest Classifier -- You can adjust parameters
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Fitting Random Forest to the Training set
rf_classifier.fit(X_rf_train, y_rf_train)

# Predicting the Test set results
y_pred_rf = rf_classifier.predict(X_rf_test)

accuracy_rf = accuracy_score(y_rf_test, y_pred_rf)
conf_matrix_rf = confusion_matrix(y_rf_test, y_pred_rf)
class_report_rf = classification_report(y_rf_test, y_pred_rf)

# Evaluate the model
print("\nRandom Forest (single) prediction results:", "\n")
print(f"Accuracy: {round(accuracy_rf*100,2)} %", "\n")
print("Confusion Matrix:")
print(conf_matrix_rf, "\n")
print("Classification Report:")
print(class_report_rf)
```

```
Random Forest (single) prediction results:

Accuracy: 72.13 %

Confusion Matrix:
[[11906  5307]
 [ 4297 12947]]

Classification Report:
              precision    recall  f1-score   support

       False       0.73      0.69      0.71     17213
        True       0.71      0.75      0.73     17244

    accuracy                           0.72     34457
   macro avg       0.72      0.72      0.72     34457
weighted avg       0.72      0.72      0.72     34457
```

Confusion Matrix Random Forest Model

Entrée [60]:
```
# Plot confusion matrix
graph_confusion_matrix = ConfusionMatrixDisplay.from_predictions(y_log_test, y_pred_log)
```



Cross-Validation of Models

In this step, we conduct cross-validation on the logistic regression and Random Forest models to assess their reliability across different subsets of the dataset. Utilizing the KFold method with 5 splits and shuffling enabled, the dataset is partitioned into distinct subsets for multiple training and testing cycles. The average of these scores is then calculated and displayed, providing a robust measure of the models' overall performance. This approach helps to ensure that the predictive accuracy of the models is not overly influenced by any specific partition of the data, thereby increasing confidence in their generalizability.

Entrée [61]:
```
# Verify model by averaging different test/train splits
k_folds = KFold(n_splits=5, shuffle=True)
# The number of folds determines the test/train split for each iteration.
# So 5 folds has 5 different mutually exclusive training sets.
# That's a 1 to 4 (or .20 to .80) testing/training split for each of the 5 iterations.

# This is the average score. Print 'scores' to see array of individual iteration scores.
log_scores = cross_val_score(mylog_model, X_log, y_log)
rf_scores = cross_val_score(rf_classifier, X_rf, y_rf)

# Output average scores
print(
    "Logistic Regression Average Prediction Score: ",
    round(log_scores.mean() * 100, 2),
    "%",
)
print("Random Forest Average Prediction Score: ", round(rf_scores.mean() * 100, 2), "%")
```

```
Logistic Regression Average Prediction Score:  74.26 %
Random Forest Average Prediction Score:  71.89 %
```

Patient Outcome Prediction

The primary objective of this project is to predict whether a patient is at risk of diabetes. This is achieved through the integration of an interactive interface where users input various health indicators. Subsequently, the system generates predictions based on the provided inputs, aiding in proactive healthcare management and risk assessment for diabetes.

Entrée [62]:
```
# USER INTERFACE: form for input for patient prediction
import ipywidgets as widgets

# 'features_dict' is a dictionary mapping feature to description
features_dict = {
    "Sex": "Sex:",
    "Age": "Age category (1 = 18-24, 13 = 80 or older; see table):",
    "BMI": "Body Mass Index:",
    "HighBP": "High Blood Pressure",
    "HighChol": "High Cholesterol",
    "Smoker": "Have you smoked at least 100 cigarettes in your life?",
    "HvyAlcoholConsump": "Heavy drinkers (drinks <14 for men, <7 for women per week",
    "Stroke": "(Ever told) you had a Stroke?",
    "HeartDiseaseorAttack": "Heart Disease or Attack (CHD or MI)",
    "GenHlth": "General Health scale :",
    "MentHlth": "How many past days was your Mental Health not good?",
    "PhysHlth": "How many past days was your Physical Health not good?",
    "DiffWalk": "Do you have Difficulty Walking or climbing stairs?",
    "PhysActivity": "Physical Activity in past 30 days, not incl job",
    "Fruits": "Eat 1 Fruit or more per day",
    "Veggies": "Eat Veggies 1 or more per day",
}
widgets_dict = {}

# Create widgets for each feature
for item in features_dict.keys():
    if item in [
        "HighBP",
        "HighChol",
        "Smoker",
        "Stroke",
        "HeartDiseaseorAttack",
        "PhysActivity",
        "Fruits",
        "Veggies",
        "HvyAlcoholConsump",
        "DiffWalk",
    ]:
        # Binary features: create a dropdown with options 'Yes' and 'No'
        widgets_dict[item + "_label"] = widgets.Label(
            features_dict.get(item), layout={"width": "max-content"}
        )
        widgets_dict[item] = widgets.RadioButtons(
            options={"No": 0, "Yes": 1},
            value=0,
        )
    if item in ["Sex"]:
        # Create a dropdown with options 'Male' and 'Female'
        widgets_dict[item + "_label"] = widgets.Label(
            features_dict.get(item), layout={"width": "max-content"}
        )
        widgets_dict[item] = widgets.Dropdown(
            options=[("Female", 0), ("Male", 1)],
            value=0,
        )
    # Numerical features: create float sliders
    if item in ["BMI"]:
        widgets_dict[item + "_label"] = widgets.Label(
            features_dict.get(item), layout={"width": "max-content"}
        )
        widgets_dict[item] = widgets.FloatSlider(
            value=20.0,
            min=10,
            max=50.0,
            step=0.1,
        )
    if item in ["GenHlth"]:
        widgets_dict[item + "_label"] = widgets.Label(
            features_dict.get(item), layout={"width": "max-content"}
        )
        widgets_dict[item] = widgets.FloatSlider(
            value=3,
            min=1,
            max=5,
            step=1,
        )
    if item in ["MentHlth", "PhysHlth"]:
        widgets_dict[item + "_label"] = widgets.Label(
            features_dict.get(item), layout={"width": "max-content"}
        )
        widgets_dict[item] = widgets.FloatSlider(
            value=0,
            min=0,
            max=30,
            step=1,
        )
    if item in ["Age"]:
        widgets_dict[item + "_label"] = widgets.Label(
            features_dict.get(item), layout={"width": "max-content"}
        )
        widgets_dict[item] = widgets.FloatSlider(
            value=8,
```

```
            min=1,
            max=13,
            step=1,
        )

# Button to make prediction
predict_btn = widgets.Button(description="Predict Patient Risk")

# Output widget to display prediction result
output = widgets.Output()


def on_predict_btn_clicked(b):
    # Prepare the input for the model
    input_data = [widgets_dict[feature].value for feature in features_dict.keys()]
    input_data = np.array(input_data).reshape(1, -1)

    # Create a DataFrame with input_data and assign column names using features
    input_df = pd.DataFrame(input_data, columns=features_dict.keys())

    # Ensure the DataFrame columns are in the correct order
    input_df = input_df[X_log.columns]

    # Apply the same scaling to the input as was done to the training data
    input_df[numerical_columns] = scaler.transform(input_df[numerical_columns])

    #### Make prediction ####
    prediction = mylog_model.predict(input_df)

    # Display prediction
    with output:
        output.clear_output()
        if prediction[0] == 0:
            print("Prediction: Not at risk of diabetes")
        else:
            print("Prediction: At risk of diabetes")


predict_btn.on_click(on_predict_btn_clicked)

# Display widgets
for widget in widgets_dict.values():
    display(widget)
display(predict_btn, output)
```

Sex:

Female

Age category (1 = 18-24, 13 = 80 or older; see table):

8.00

Body Mass Index:

20.00

High Blood Pressure

- (•) No
- ( ) Yes

High Cholesterol

- (•) No
- ( ) Yes

Have you smoked at least 100 cigarettes in your life?

- (•) No
- ( ) Yes

Heavy drinkers (drinks <14 for men, <7 for women per week

- (•) No
- ( ) Yes

(Ever told) you had a Stroke?

- (•) No
- ( ) Yes

Heart Disease or Attack (CHD or MI)

- (•) No
- ( ) Yes

General Health scale :

3.00

How many past days was your Mental Health not good?

0.00

How many past days was your Physical Health not good?

0.00

Do you have Difficulty Walking or climbing stairs?

- (•) No
- ( ) Yes

Physical Activity in past 30 days, not incl job

- (•) No
- ( ) Yes

Eat 1 Fruit or more per day

- (•) No
- ( ) Yes

Eat Veggies 1 or more per day

- (•) No
- ( ) Yes

Predict Patient Risk