health-indicators

July 27, 2024

0.0.1 Health Indication with three prediction models

[1]:
```python
import os
os.environ['LANG'] = 'en_US.UTF-8'
os.environ['LC_ALL'] = 'en_US.UTF-8'

# Now install the necessary packages
!pip install shap
```

```
Requirement already satisfied: shap in c:\users\navee\anaconda3\lib\site-
packages (0.46.0)
Requirement already satisfied: numpy in c:\users\navee\anaconda3\lib\site-
packages (from shap) (1.23.3)
Requirement already satisfied: scipy in c:\users\navee\anaconda3\lib\site-
packages (from shap) (1.11.4)
Requirement already satisfied: scikit-learn in
c:\users\navee\anaconda3\lib\site-packages (from shap) (1.2.2)
Requirement already satisfied: pandas in c:\users\navee\anaconda3\lib\site-
packages (from shap) (2.1.4)
Requirement already satisfied: tqdm>=4.27.0 in
c:\users\navee\anaconda3\lib\site-packages (from shap) (4.65.0)
Requirement already satisfied: packaging>20.9 in
c:\users\navee\anaconda3\lib\site-packages (from shap) (23.1)
Requirement already satisfied: slicer==0.0.8 in
c:\users\navee\anaconda3\lib\site-packages (from shap) (0.0.8)
Requirement already satisfied: numba in c:\users\navee\anaconda3\lib\site-
packages (from shap) (0.59.0)
Requirement already satisfied: cloudpickle in c:\users\navee\anaconda3\lib\site-
packages (from shap) (2.2.1)
Requirement already satisfied: colorama in c:\users\navee\anaconda3\lib\site-
packages (from tqdm>=4.27.0->shap) (0.4.6)
Requirement already satisfied: llvmlite<0.43,>=0.42.0dev0 in
c:\users\navee\anaconda3\lib\site-packages (from numba->shap) (0.42.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
c:\users\navee\anaconda3\lib\site-packages (from pandas->shap) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
c:\users\navee\anaconda3\lib\site-packages (from pandas->shap) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in
```

```
c:\users\navee\anaconda3\lib\site-packages (from pandas->shap) (2023.3)
Requirement already satisfied: joblib>=1.1.1 in
c:\users\navee\anaconda3\lib\site-packages (from scikit-learn->shap) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
c:\users\navee\anaconda3\lib\site-packages (from scikit-learn->shap) (2.2.0)
Requirement already satisfied: six>=1.5 in c:\users\navee\anaconda3\lib\site-
packages (from python-dateutil>=2.8.2->pandas->shap) (1.16.0)
```

[2]:
```
# !git clone https://github.com/rapidsai/rapidsai-csp-utils.git
# !python rapidsai-csp-utils/colab/pip-install.py
```

[3]:
```
!pip install numpy==1.23.3
```

```
Requirement already satisfied: numpy==1.23.3 in
c:\users\navee\anaconda3\lib\site-packages (1.23.3)
```

[4]:
```
!pip install imblearn
```

```
Requirement already satisfied: imblearn in c:\users\navee\anaconda3\lib\site-
packages (0.0)
Requirement already satisfied: imbalanced-learn in
c:\users\navee\anaconda3\lib\site-packages (from imblearn) (0.11.0)
Requirement already satisfied: numpy>=1.17.3 in
c:\users\navee\anaconda3\lib\site-packages (from imbalanced-learn->imblearn)
(1.23.3)
Requirement already satisfied: scipy>=1.5.0 in
c:\users\navee\anaconda3\lib\site-packages (from imbalanced-learn->imblearn)
(1.11.4)
Requirement already satisfied: scikit-learn>=1.0.2 in
c:\users\navee\anaconda3\lib\site-packages (from imbalanced-learn->imblearn)
(1.2.2)
Requirement already satisfied: joblib>=1.1.1 in
c:\users\navee\anaconda3\lib\site-packages (from imbalanced-learn->imblearn)
(1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
c:\users\navee\anaconda3\lib\site-packages (from imbalanced-learn->imblearn)
(2.2.0)
```

[5]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier as rf
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split, GridSearchCV
# import cuml
```

By looking at the dataset, I will visualize the relationship between the lifestyle of a person and diabetes. With lifestyle features as Smoker, Physical Activities, Fruits, HealthTracker, Income factors; this dataset is trying to predict if the lifestyle of a person can impact the health of the patient or not. It can help to identify what key lifestyle indicators can influence diabetes in a person and can help healthcare providers to intervene early to prevent the onset of diabetes.

1 Prepare the Data

[6]:
```
df = pd.read_csv('diabetes_012_health_indicators_BRFSS2015.csv')
df.head()
```

[6]:
```
   Diabetes_012  HighBP  HighChol  CholCheck   BMI  Smoker  Stroke  \
0           0.0     1.0       1.0        1.0  40.0     1.0     0.0
1           0.0     0.0       0.0        0.0  25.0     1.0     0.0
2           0.0     1.0       1.0        1.0  28.0     0.0     0.0
3           0.0     1.0       0.0        1.0  27.0     0.0     0.0
4           0.0     1.0       1.0        1.0  24.0     0.0     0.0

   HeartDiseaseorAttack  PhysActivity  Fruits  …  AnyHealthcare  \
0                   0.0           0.0     0.0  …            1.0
1                   0.0           1.0     0.0  …            0.0
2                   0.0           0.0     1.0  …            1.0
3                   0.0           1.0     1.0  …            1.0
4                   0.0           1.0     1.0  …            1.0

   NoDocbcCost  GenHlth  MentHlth  PhysHlth  DiffWalk  Sex   Age  Education  \
0          0.0      5.0      18.0      15.0       1.0  0.0   9.0        4.0
1          1.0      3.0       0.0       0.0       0.0  0.0   7.0        6.0
2          1.0      5.0      30.0      30.0       1.0  0.0   9.0        4.0
3          0.0      2.0       0.0       0.0       0.0  0.0  11.0        3.0
4          0.0      2.0       3.0       0.0       0.0  0.0  11.0        5.0

   Income
0     3.0
1     1.0
2     8.0
3     6.0
4     4.0

[5 rows x 22 columns]
```

[7]:
```
df.describe()
```

[7]:
```
        Diabetes_012         HighBP       HighChol      CholCheck  \
count  253680.000000  253680.000000  253680.000000  253680.000000
mean        0.296921       0.429001       0.424121       0.962670
std         0.698160       0.494934       0.494210       0.189571
min         0.000000       0.000000       0.000000       0.000000
```

```
25%          0.000000       0.000000       0.000000       1.000000
50%          0.000000       0.000000       0.000000       1.000000
75%          0.000000       1.000000       1.000000       1.000000
max          2.000000       1.000000       1.000000       1.000000

                  BMI         Smoker         Stroke  HeartDiseaseorAttack  \
count   253680.000000  253680.000000  253680.000000         253680.000000
mean        28.382364       0.443169       0.040571              0.094186
std          6.608694       0.496761       0.197294              0.292087
min         12.000000       0.000000       0.000000              0.000000
25%         24.000000       0.000000       0.000000              0.000000
50%         27.000000       0.000000       0.000000              0.000000
75%         31.000000       1.000000       0.000000              0.000000
max         98.000000       1.000000       1.000000              1.000000

         PhysActivity         Fruits  …  AnyHealthcare    NoDocbcCost  \
count   253680.000000  253680.000000  …  253680.000000  253680.000000
mean         0.756544       0.634256  …       0.951053       0.084177
std          0.429169       0.481639  …       0.215759       0.277654
min          0.000000       0.000000  …       0.000000       0.000000
25%          1.000000       0.000000  …       1.000000       0.000000
50%          1.000000       1.000000  …       1.000000       0.000000
75%          1.000000       1.000000  …       1.000000       0.000000
max          1.000000       1.000000  …       1.000000       1.000000

              GenHlth       MentHlth       PhysHlth       DiffWalk  \
count   253680.000000  253680.000000  253680.000000  253680.000000
mean         2.511392       3.184772       4.242081       0.168224
std          1.068477       7.412847       8.717951       0.374066
min          1.000000       0.000000       0.000000       0.000000
25%          2.000000       0.000000       0.000000       0.000000
50%          2.000000       0.000000       0.000000       0.000000
75%          3.000000       2.000000       3.000000       0.000000
max          5.000000      30.000000      30.000000       1.000000

                  Sex            Age      Education         Income
count   253680.000000  253680.000000  253680.000000  253680.000000
mean         0.440342       8.032119       5.050434       6.053875
std          0.496429       3.054220       0.985774       2.071148
min          0.000000       1.000000       1.000000       1.000000
25%          0.000000       6.000000       4.000000       5.000000
50%          0.000000       8.000000       5.000000       7.000000
75%          1.000000      10.000000       6.000000       8.000000
max          1.000000      13.000000       6.000000       8.000000

[8 rows x 22 columns]
```

[8]:
```
#Checking the data type of each column
df.dtypes
```

[8]:
```
Diabetes_012            float64
HighBP                  float64
HighChol                float64
CholCheck               float64
BMI                     float64
Smoker                  float64
Stroke                  float64
HeartDiseaseorAttack    float64
PhysActivity            float64
Fruits                  float64
Veggies                 float64
HvyAlcoholConsump       float64
AnyHealthcare           float64
NoDocbcCost             float64
GenHlth                 float64
MentHlth                float64
PhysHlth                float64
DiffWalk                float64
Sex                     float64
Age                     float64
Education               float64
Income                  float64
dtype: object
```

[9]:
```
#Value counts of our target_variable
df['Diabetes_012'].value_counts()
```

[9]:
```
Diabetes_012
0.0    213703
2.0     35346
1.0      4631
Name: count, dtype: int64
```

[10]:
```
df['GenHlth'].value_counts()
```

[10]:
```
GenHlth
2.0    89084
3.0    75646
1.0    45299
4.0    31570
5.0    12081
Name: count, dtype: int64
```

[11]:
```
#Converting each datatype into its correct datatype
#categorical variable
categorical_columns = ['Diabetes_012','Age', 'Education', 'Income', 'GenHlth']
#Rest of the columns (binary indicators)
integer_columns = df.columns.difference(categorical_columns)
#continuous variable
float_columns = ['BMI','MentHlth', 'PhysHlth']
integer_columns=integer_columns.drop(float_columns)
```

[12]:
```
#converting into desired types:
df[categorical_columns] = df[categorical_columns].astype('int64').
  ↪astype('category')
df[integer_columns] = df[integer_columns].astype('int64')
df[float_columns] = df[float_columns].astype('float64')
```

[13]:
```
df.dtypes
```

[13]:
```
Diabetes_012            category
HighBP                     int64
HighChol                   int64
CholCheck                  int64
BMI                      float64
Smoker                     int64
Stroke                     int64
HeartDiseaseorAttack       int64
PhysActivity               int64
Fruits                     int64
Veggies                    int64
HvyAlcoholConsump          int64
AnyHealthcare              int64
NoDocbcCost                int64
GenHlth                 category
MentHlth                 float64
PhysHlth                 float64
DiffWalk                   int64
Sex                        int64
Age                     category
Education               category
Income                  category
dtype: object
```

CHECK FOR MISSING VALUES:

[14]:
```
df.isnull().sum()
```

[14]:
```
Diabetes_012            0
HighBP                  0
```

```
HighChol                0
CholCheck               0
BMI                     0
Smoker                  0
Stroke                  0
HeartDiseaseorAttack    0
PhysActivity            0
Fruits                  0
Veggies                 0
HvyAlcoholConsump       0
AnyHealthcare           0
NoDocbcCost             0
GenHlth                 0
MentHlth                0
PhysHlth                0
DiffWalk                0
Sex                     0
Age                     0
Education               0
Income                  0
dtype: int64
```

There's no missing values. Let's move on with invalid values in the dataset.

[15]:
```
#BINARY CHECK
df[integer_columns].apply(lambda x: ((x != 0) & (x != 1)).sum())
```

[15]:
```
AnyHealthcare           0
CholCheck               0
DiffWalk                0
Fruits                  0
HeartDiseaseorAttack    0
HighBP                  0
HighChol                0
HvyAlcoholConsump       0
NoDocbcCost             0
PhysActivity            0
Sex                     0
Smoker                  0
Stroke                  0
Veggies                 0
dtype: int64
```

[16]:
```
#CATEGORY CHECK FOR CATEGORICAL COLUMNS
for i in categorical_columns:
  print(df[i].value_counts())
```

```
Diabetes_012
```

```
0    213703
2     35346
1      4631
Name: count, dtype: int64
Age
9     33244
10    32194
8     30832
7     26314
11    23533
6     19819
13    17363
5     16157
12    15980
4     13823
3     11123
2      7598
1      5700
Name: count, dtype: int64
Education
6    107325
5     69910
4     62750
3      9478
2      4043
1       174
Name: count, dtype: int64
Income
8    90385
7    43219
6    36470
5    25883
4    20135
3    15994
2    11783
1     9811
Name: count, dtype: int64
GenHlth
2    89084
3    75646
1    45299
4    31570
5    12081
Name: count, dtype: int64
```

[17]:

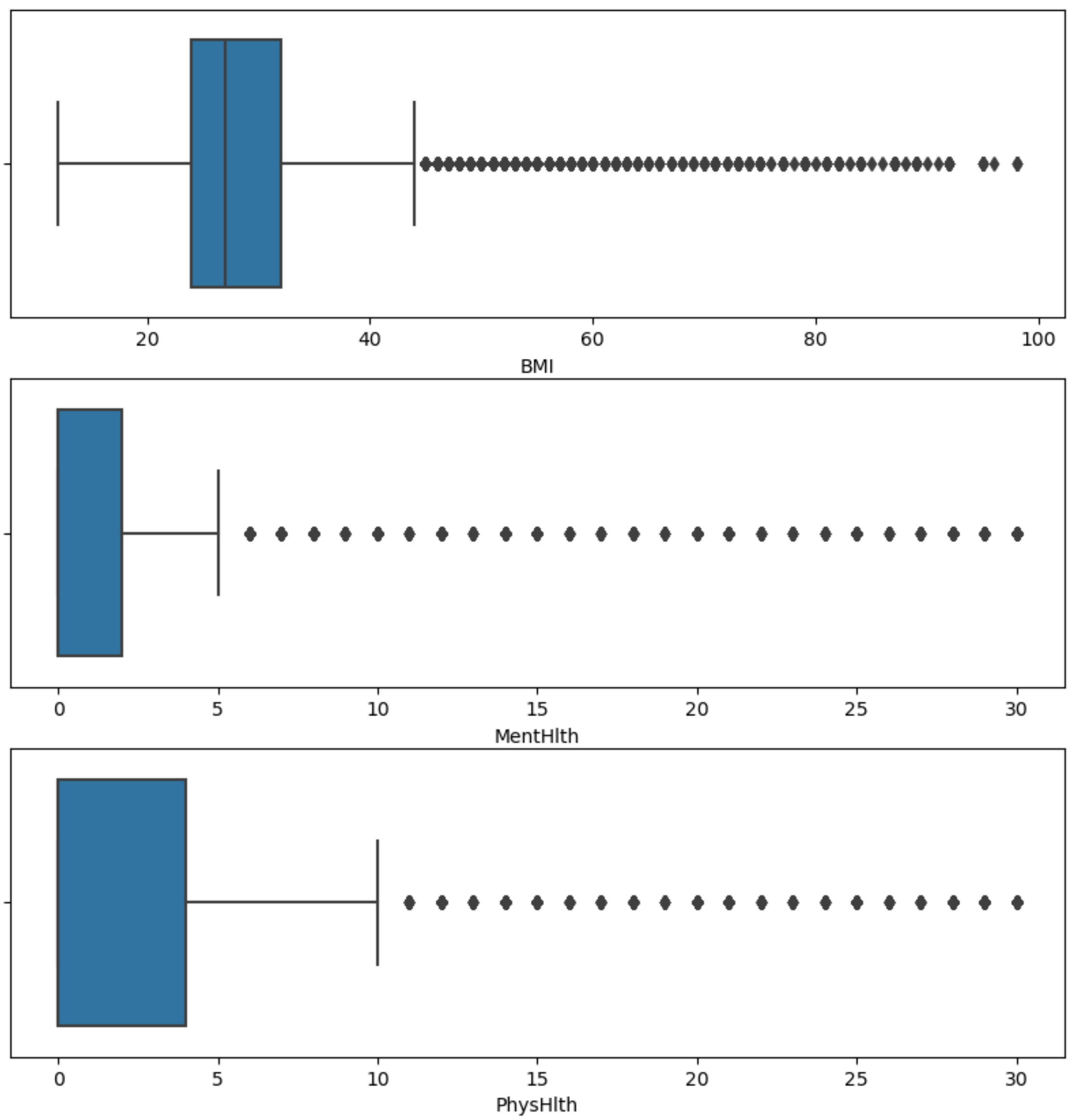
```python
#Checking for duplicates
df.duplicated().sum()
```

[17]: 23899

[18]:
```
#dropping duplicates
df.drop_duplicates(inplace=True)
```

There's no invalid value in the dataset Outlier check

[19]:
```
#Checking the outlier through box plot of float columns
fig, ax = plt.subplots(3,1, figsize=(10,10))
for i,j in enumerate(float_columns):
  if j =='BMI':
    sns.boxplot(x=(df[j]), ax=ax[i])
  else:
    sns.boxplot(x=(df[j]), ax=ax[i])
```



1.0.1 Outlier Analysis

1. BMI: Several points are far beyond the typical range, which could indicate measurement errors or extreme cases.

2. MentHlth and PhysHlth: These represent days affected by mental or physical health issues respectively. The outliers suggest that some respondents reported the maximum possible days, which might be true extremes or data entry exaggerations.

[20]:
```
len(df[(df['PhysHlth']>10) & (df['Diabetes_012'].isin([1,2]) & (df['BMI']>40))])
```

[20]: 1690

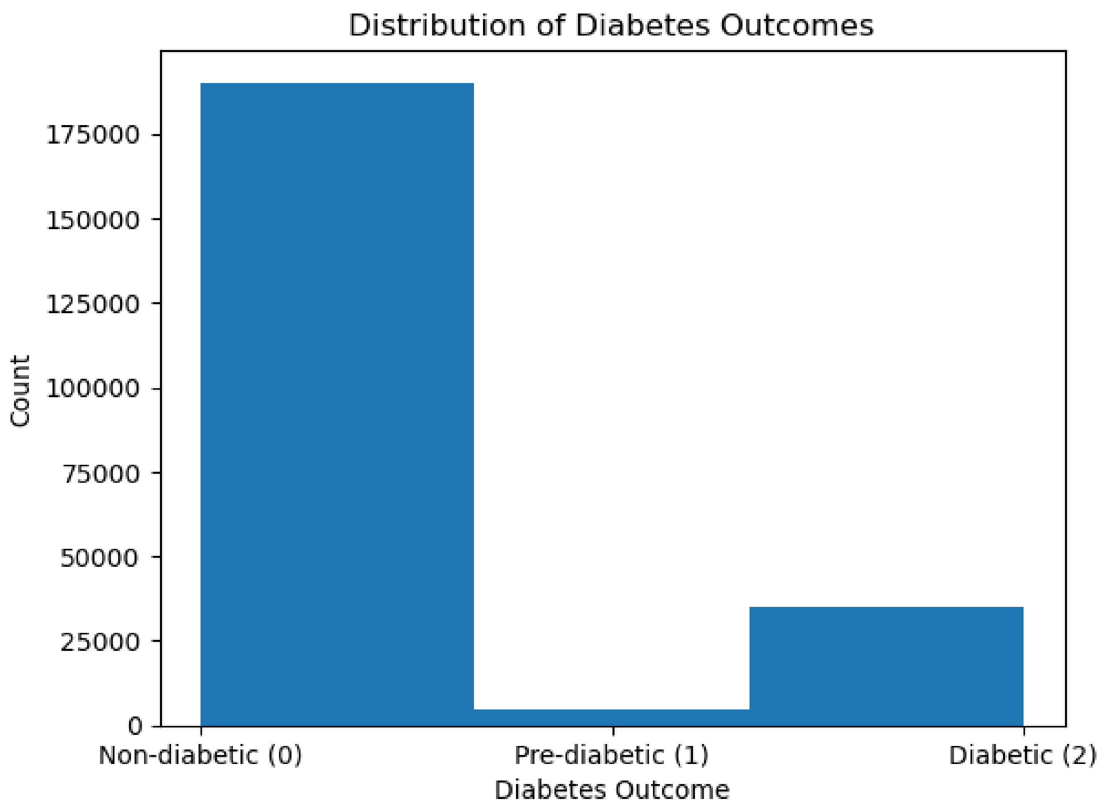
[21]:
```
len(df[(df['PhysHlth']>10) & (df['Diabetes_012'].isin([0]) & (df['BMI']>40))])
```

[21]: 1614

We will plan to scale these continous variables to handle outliers. Removing the outliers can impact the class record for person having diabetes because person having higher BMI, PhysHlth, MentHlth

Let's analyze our class distribution

[22]:
```
#Distribution of our target variable through histogram
plt.hist(df['Diabetes_012'], bins=3)
plt.title('Distribution of Diabetes Outcomes')
plt.xlabel('Diabetes Outcome')
plt.ylabel('Count')
plt.xticks([0,1,2], ['Non-diabetic (0)', 'Pre-diabetic (1)', 'Diabetic (2)'])
plt.show()
```

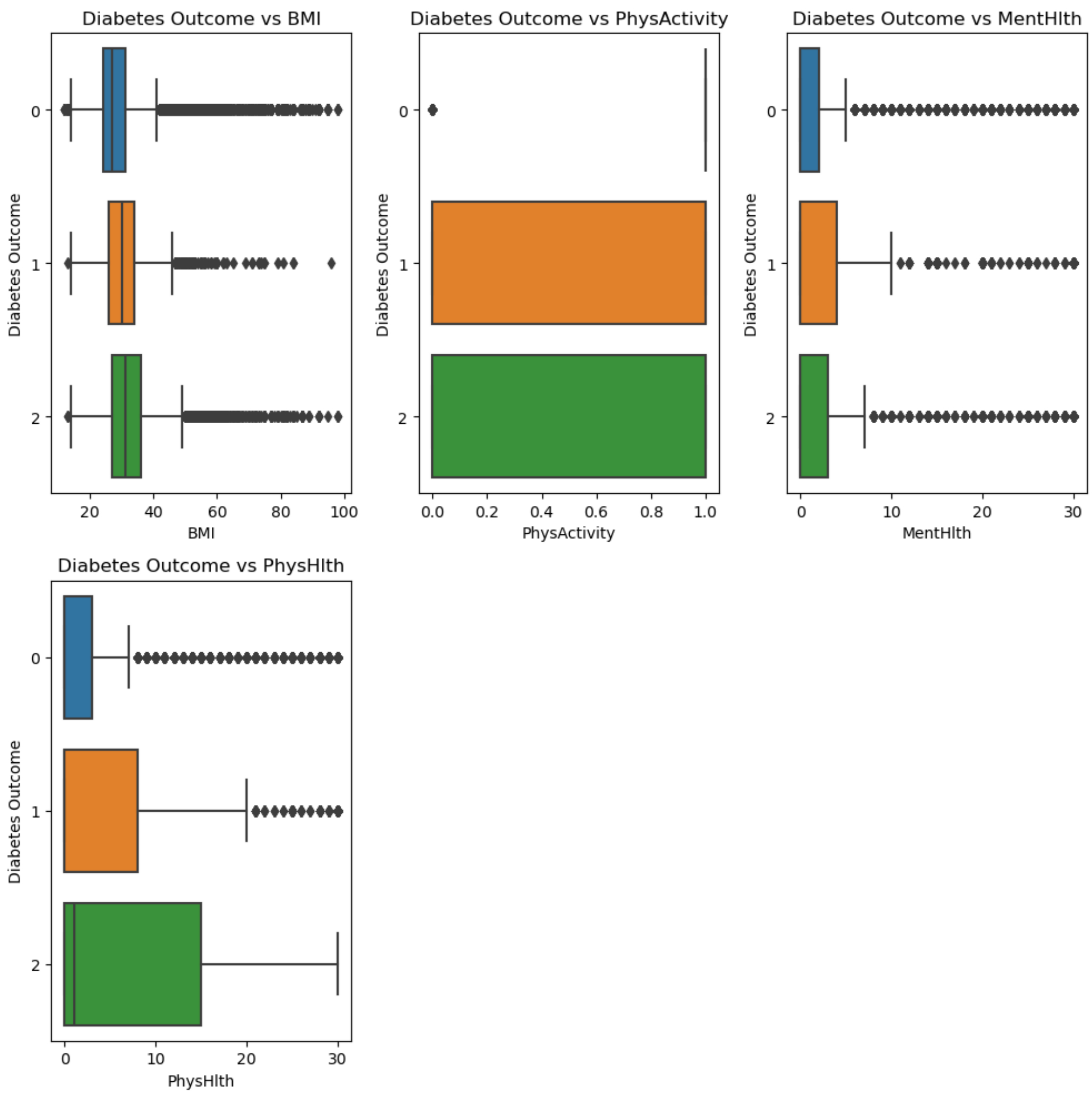


There's a huge class imbalance and would require some sort of sampling or assigning class weights to minority class to avoid Biasness towards Non-diabetic person. Let's see the relationship of Diabetes_012 column with other variables

[23]:
```
significant_vars = ['BMI', 'PhysActivity', 'MentHlth', 'PhysHlth']
plt.figure(figsize=(10, 10))
for i, var in enumerate(significant_vars):
    plt.subplot(2, 3, i+1)
    if var =='Age':
      #horizontal plot
      sns.barplot(x=df[var], y=df['Diabetes_012'],orient = 'h')
    else:
      sns.boxplot(x=df[var], y=df['Diabetes_012'])
    plt.title(f'Diabetes Outcome vs {var}')
    plt.ylabel('Diabetes Outcome')
    plt.xlabel(var)

plt.tight_layout()
plt.show()
```

```
C:\Users\navee\anaconda3\Lib\site-packages\seaborn\categorical.py:641:
FutureWarning: The default of observed=False is deprecated and will be changed
to True in a future version of pandas. Pass observed=False to retain current
behavior or observed=True to adopt the future default and silence this warning.
  grouped_vals = vals.groupby(grouper)
C:\Users\navee\anaconda3\Lib\site-packages\seaborn\categorical.py:641:
FutureWarning: The default of observed=False is deprecated and will be changed
to True in a future version of pandas. Pass observed=False to retain current
behavior or observed=True to adopt the future default and silence this warning.
  grouped_vals = vals.groupby(grouper)
C:\Users\navee\anaconda3\Lib\site-packages\seaborn\categorical.py:641:
FutureWarning: The default of observed=False is deprecated and will be changed
to True in a future version of pandas. Pass observed=False to retain current
behavior or observed=True to adopt the future default and silence this warning.
  grouped_vals = vals.groupby(grouper)
C:\Users\navee\anaconda3\Lib\site-packages\seaborn\categorical.py:641:
FutureWarning: The default of observed=False is deprecated and will be changed
to True in a future version of pandas. Pass observed=False to retain current
behavior or observed=True to adopt the future default and silence this warning.
  grouped_vals = vals.groupby(grouper)
```



Analysis

1. BMI vs. Diabetes Outcome: Higher BMI values tend to cluster more around higher diabetes outcomes (pre-diabetic and diabetic). This suggests a potential correlation where higher BMI might be associated with an increased risk of diabetes.

2. Age vs. Diabetes Outcome: Older ages appear to have a higher concentration of diabetic outcomes, indicating age as a significant risk factor.

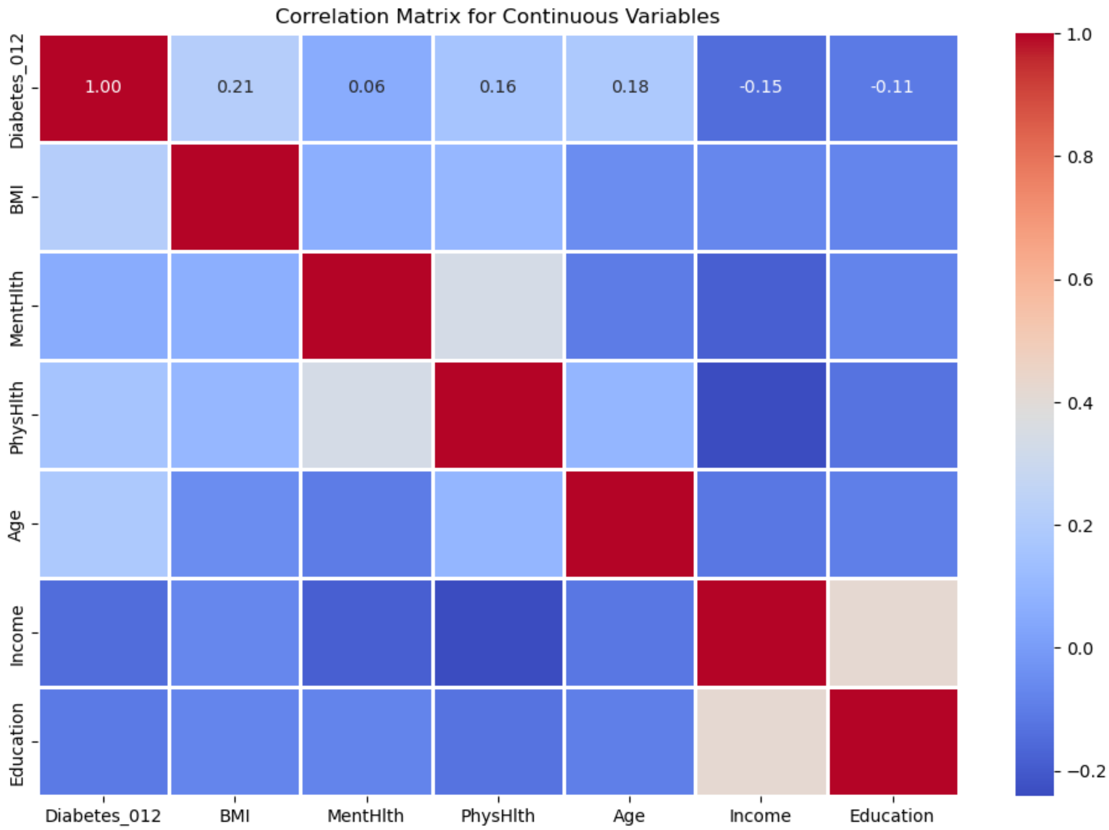
3. Physical Activity (PhysActivity) vs. Diabetes Outcome: Lower levels of physical activity correspond to higher diabetes outcomes, supporting the hypothesis that physical inactivity may be associated with an increased risk of diabetes.

4. **Mental Health (MentHlth) and Physical Health (PhysHlth) vs. Diabetes Outcome: Both show varied distributions, suggesting that worse health scores (both mental and physical) might be associated with higher diabetes outcomes.**

2 Performing significance tests to determine if the patterns that are detected above are statistically significant.

2.0.1 Correlation Matrix with continuous columns

[24]:
```
continuous_columns = ['BMI', 'MentHlth', 'PhysHlth','Age','Income','Education']
#log_columns = ['BMI', 'MentHlth', 'PhysHlth']
correlation_matrix = df[['Diabetes_012'] + continuous_columns]
#correlation_matrix[log_columns] = np.log(correlation_matrix[log_columns])
correlation_matrix=correlation_matrix.corr()
# Visualize the correlation matrix using a heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f",␣
↪linewidths=1)
plt.title('Correlation Matrix for Continuous Variables')
plt.show()
```



Analysis

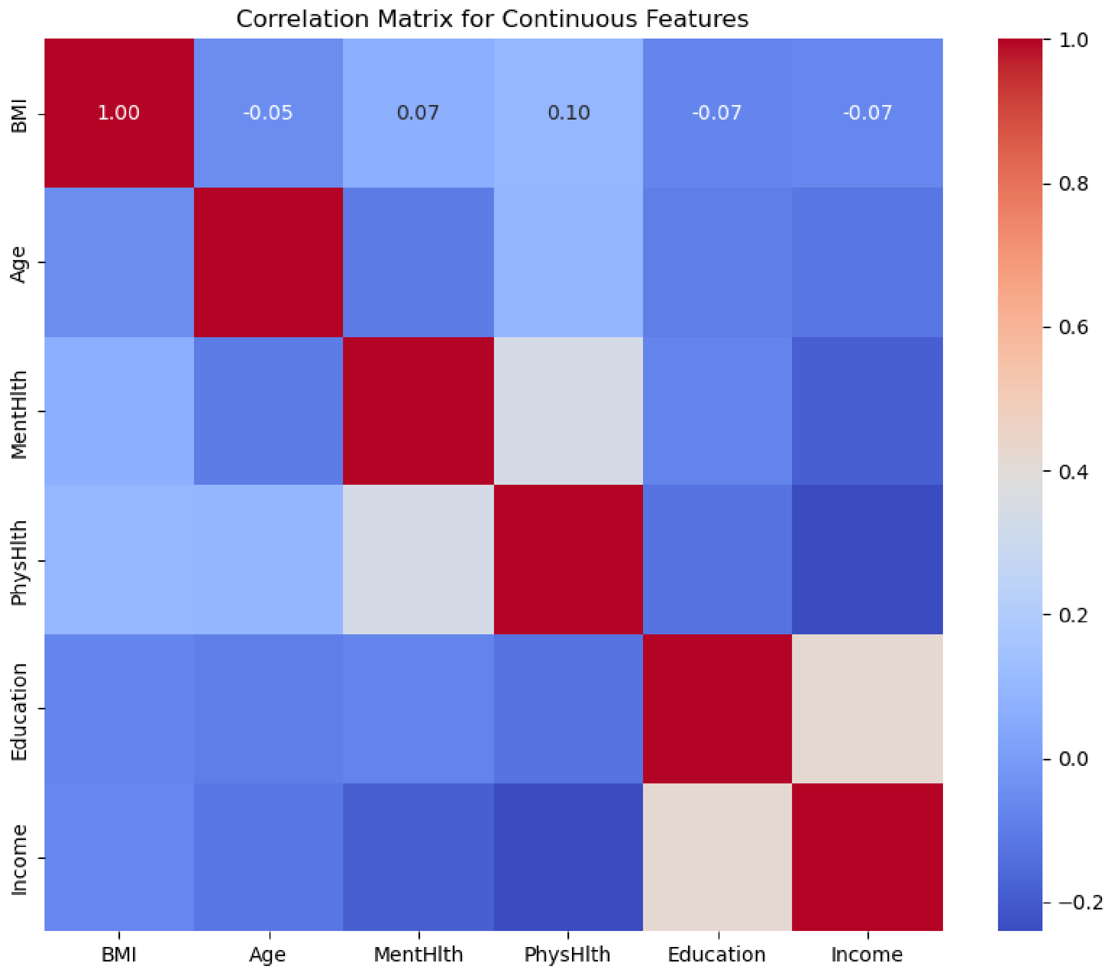
1. BMI shows a moderate positive correlation with diabetes outcomes, suggesting that as BMI increases, so does the likelihood of being diabetic or pre-diabetic.

2. Age also has a moderate positive correlation with diabetes outcomes, indicating that older individuals have a higher likelihood of having diabetes.

3. Income and Education show a slight negative correlation with diabetes outcomes, hinting that higher income and education levels may be associated with lower diabetes prevalence. Multicollinearilty test with the variables

[25]:
```
correlation_matrix = df[['BMI', 'Age', 'MentHlth', 'PhysHlth', 'Education',␣
↪'Income']].corr()

# Visualize the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix for Continuous Features')
plt.show()
```



That is a good sign to see that significant variables are not correlated with each other. (0.45 is a moderate relation)

Statistical T-test

[26]:
```
from scipy.stats import ttest_ind

# Function to perform t-test for binary categorical variables
def perform_t_test(df, binary_var, outcome_var):
    group1 = df[df[binary_var] == 0][outcome_var]
    group2 = df[df[binary_var] == 1][outcome_var]
    t_stat, p_value = ttest_ind(group1, group2)
    return p_value

# Perform t-tests for binary variables
t_test_results = {col: perform_t_test(df, col, 'Diabetes_012') for col in␣
↪integer_columns}
```

```
t_test_results
```

[26]:
```
{'AnyHealthcare': 7.076078548666687e-33,
 'CholCheck': 3.824009404772498e-289,
 'DiffWalk': 0.0,
 'Fruits': 2.8395711130029448e-34,
 'HeartDiseaseorAttack': 0.0,
 'HighBP': 0.0,
 'HighChol': 0.0,
 'HvyAlcoholConsump': 6.350431787393671e-228,
 'NoDocbcCost': 1.3270340932366845e-29,
 'PhysActivity': 0.0,
 'Sex': 6.483156712626619e-54,
 'Smoker': 1.8493199150591577e-111,
 'Stroke': 0.0,
 'Veggies': 2.0592080730308895e-96}
```

2.0.2 Since the p-values of all the variables are coming out to be very low and given that the variables semantic meaning shows a good relation with the target variable, it seems fair to use all.

2.0.3 Feature Selection:

[27]:
```
#Feature Engineering: Combining Fruits and Veggies by naming it as Diet Index.
df['DietIntake'] = df['Fruits'] + df['Veggies']
#Performing t-test with target variable to check its significance
perform_t_test(df, 'DietIntake', 'Diabetes_012')
```

[27]:
```
6.250069453978259e-13
```

Value is pretty small, we can use it instead of Fruits and Veggies

[28]:
```
from scipy.stats import chi2_contingency

# Function to perform chi-square test and return p-value
def perform_chi_square_test(data, categorical_var, outcome_var):
    contingency_table = pd.crosstab(data[categorical_var], data[outcome_var])
    _, p_value, _, _ = chi2_contingency(contingency_table)
    return p_value

# Perform chi-square tests for 'Education' and 'Income'
chi_square_results = {
    "Education": perform_chi_square_test(df, 'Education', 'Diabetes_012'),
    "Income": perform_chi_square_test(df, 'Income', 'Diabetes_012'),
    "Age": perform_chi_square_test(df, 'Age', 'Diabetes_012'),
    "GenHlth": perform_chi_square_test(df, 'GenHlth', 'Diabetes_012'),
}
```

```
chi_square_results
```

[28]:
```
{'Education': 0.0, 'Income': 0.0, 'Age': 0.0, 'GenHlth': 0.0}
```

[29]:
```
#Final feature selection from correlation and t-test
final_features = ['BMI', 'Age', 'PhysHlth', 'Income','Diabetes_012','MentHlth']␣
 ↪#correlation
#Based on t-test
final_features.
 ↪extend(['CholCheck','DiffWalk','HeartDiseaseorAttack','HighBP','HighChol','HvyAlcoholConsum
```

[30]:
```
df[final_features]
```

[30]:
```
         BMI  Age  PhysHlth  Income  Diabetes_012  MentHlth  CholCheck  DiffWalk  \
0       40.0    9      15.0       3             0      18.0          1         1
1       25.0    7       0.0       1             0       0.0          0         0
2       28.0    9      30.0       8             0      30.0          1         1
3       27.0   11       0.0       6             0       0.0          1         0
4       24.0   11       0.0       4             0       3.0          1         0
…        …   ..       …     …          …        …        …        …
253675  45.0    5       5.0       7             0       0.0          1         0
253676  18.0   11       0.0       4             2       0.0          1         1
253677  28.0    2       0.0       2             0       0.0          1         0
253678  23.0    7       0.0       1             0       0.0          1         0
253679  25.0    9       0.0       2             2       0.0          1         0

        HeartDiseaseorAttack  HighBP  HighChol  HvyAlcoholConsump  \
0                          0       1         1                  0
1                          0       0         0                  0
2                          0       1         1                  0
3                          0       1         0                  0
4                          0       1         1                  0
…                        …     …       …                 …
253675                     0       1         1                  0
253676                     0       1         1                  0
253677                     0       0         0                  0
253678                     0       1         0                  0
253679                     1       1         1                  0

        NoDocbcCost  PhysActivity  Sex  Smoker  Stroke  DietIntake
0                 0             0    0       1       0           1
1                 1             1    0       1       0           0
2                 1             0    0       0       0           1
3                 0             1    0       0       0           2
4                 0             1    0       0       0           2
…             …             …  …     …     …          …
253675            0             0    1       0       0           2
```

```
253676              0               0     0        0        0          0
253677              0               1     0        0        0          1
253678              0               0     1        0        0          2
253679              0               1     0        0        0          1

[229781 rows x 18 columns]
```

2.0.4 Encode any categorical data. Ensure that categorical variables are represented correctly.

2.0.5 Normalize numeric data.

[31]:
```python
data = df[final_features]
```

[32]:
```python
data.head()
```

[32]:
```
    BMI  Age  PhysHlth Income Diabetes_012  MentHlth  CholCheck  DiffWalk  \
0  40.0    9      15.0      3            0      18.0          1         1
1  25.0    7       0.0      1            0       0.0          0         0
2  28.0    9      30.0      8            0      30.0          1         1
3  27.0   11       0.0      6            0       0.0          1         0
4  24.0   11       0.0      4            0       3.0          1         0

   HeartDiseaseorAttack  HighBP  HighChol  HvyAlcoholConsump  NoDocbcCost  \
0                     0       1         1                  0            0
1                     0       0         0                  0            1
2                     0       1         1                  0            1
3                     0       1         0                  0            0
4                     0       1         1                  0            0

   PhysActivity  Sex  Smoker  Stroke  DietIntake
0             0    0       1       0           1
1             1    0       1       0           0
2             0    0       0       0           1
3             1    0       0       0           2
4             1    0       0       0           2
```

[33]:
```python
data.dtypes
```

[33]:
```
BMI                      float64
Age                     category
PhysHlth                 float64
Income                  category
Diabetes_012            category
MentHlth                 float64
CholCheck                  int64
DiffWalk                   int64
```

```
HeartDiseaseorAttack      int64
HighBP                    int64
HighChol                  int64
HvyAlcoholConsump         int64
NoDocbcCost               int64
PhysActivity              int64
Sex                       int64
Smoker                    int64
Stroke                    int64
DietIntake                int64
dtype: object
```

[34]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline as ImblearnPipeline

# Load and prepare the data
X = data.drop('Diabetes_012', axis=1)
y = data['Diabetes_012']
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42, stratify=y)

# Specify the numeric, binary, and categorical features
numeric_features = ['BMI', 'PhysHlth']
binary_features = ['CholCheck', 'DiffWalk', 'HeartDiseaseorAttack', 'HighBP',␣
↪'HighChol', 'HvyAlcoholConsump', 'NoDocbcCost', 'PhysActivity', 'Sex',␣
↪'Smoker', 'Stroke', 'DietIntake']
categorical_features = ['Age', 'Income']

# Create a column transformer for preprocessing
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_features),
        ('cat', OneHotEncoder(), categorical_features),
        ('bin', 'passthrough', binary_features)  # Passthrough binary features
    ])
```

[ ]:

2.0.6 Random Forest Classifier

[35]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
```

[36]:
```
model = RandomForestClassifier(random_state=42)
param_grid = {'n_estimators': [100, 200], 'max_depth': [10, 20]}
grid_search = GridSearchCV(model, param_grid, cv=3, verbose =1)
grid_search.fit(X_train, y_train)

# # Evaluate the model
# predictions = grid_search.predict(X_test)
# print(classification_report(y_test, predictions))
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits
```

[36]:
```
GridSearchCV(cv=3, estimator=RandomForestClassifier(random_state=42),
             param_grid={'max_depth': [10, 20], 'n_estimators': [100, 200]},
             verbose=1)
```

[37]:
```
# Now use the best estimator from the grid search to make predictions
predictions = grid_search.predict(X_test)

# Evaluation
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.84      0.99      0.91     57017
           1       0.00      0.00      0.00      1389
           2       0.61      0.11      0.18     10529

    accuracy                           0.83     68935
   macro avg       0.48      0.36      0.36     68935
weighted avg       0.79      0.83      0.78     68935


C:\Users\navee\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are 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, msg_start, len(result))
C:\Users\navee\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are 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, msg_start, len(result))
```

```
C:\Users\navee\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are 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, msg_start, len(result))
```

[38]:
```
best_model_rf = grid_search.best_estimator_
best_score_rf = grid_search.best_score_
best_params = grid_search.best_params_
cv_results = pd.DataFrame(grid_search.cv_results_)
```

[39]:
```
print('Best Model:', best_model_rf)
print('Best Score:', round(best_score_rf,3))
print('Best Params:', best_params)
print('CV Results')
display(display(cv_results[['param_n_estimators', 'param_max_depth',␣
↪'mean_test_score', 'std_test_score', 'rank_test_score']]))
```

```
Best Model: RandomForestClassifier(max_depth=10, n_estimators=200,
random_state=42)
Best Score: 0.833
Best Params: {'max_depth': 10, 'n_estimators': 200}
CV Results
```

	param_n_estimators	param_max_depth	mean_test_score	std_test_score
0	100	10	0.833095	0.000555
1	200	10	0.833170	0.000584
2	100	20	0.828426	0.000355
3	200	20	0.829166	0.000562

	rank_test_score
0	2
1	1
2	4
3	3

```
None
```

2.0.7 KNeighborsClassifier

[40]:
```
import multiprocessing
#cuda gpu usage
n_jobs = multiprocessing.cpu_count()
```

[ ]:
```
```

[41]:
```
import warnings
```

[42]:
```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
import numpy as np

# Assuming X_train, X_test, y_train, y_test are already defined and are NumPy␣
↪arrays
X_train_np = np.asarray(X_train)
X_test_np = np.asarray(X_test)
y_train_np = np.asarray(y_train)
y_test_np = np.asarray(y_test)

# Define the model and parameter grid for KNN
knn = KNeighborsClassifier()
param_grid_knn = {
    'n_neighbors': [3, 5],
    'weights': ['uniform', 'distance']
}

# Setup GridSearchCV
grid_knn = GridSearchCV(knn, param_grid_knn, cv=3, scoring='accuracy',␣
↪n_jobs=-1)

# Fit GridSearchCV with the training data
grid_knn.fit(X_train_np, y_train_np)

# Evaluate the performance
print(classification_report(y_test_np, predictions))
```

```
C:\Users\navee\anaconda3\Lib\site-
packages\sklearn\model_selection\_search.py:952: UserWarning: One or more of the
test scores are non-finite: [nan nan nan nan]
  warnings.warn(
              precision    recall  f1-score   support

           0       0.84      0.99      0.91     57017
           1       0.00      0.00      0.00      1389
           2       0.61      0.11      0.18     10529

    accuracy                           0.83     68935
   macro avg       0.48      0.36      0.36     68935
weighted avg       0.79      0.83      0.78     68935

C:\Users\navee\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are 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, msg_start, len(result))
C:\Users\navee\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are 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, msg_start, len(result))
C:\Users\navee\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are 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, msg_start, len(result))
```

[43]:
```
best_model_knn = grid_knn.best_estimator_
best_score_knn = grid_knn.best_score_
best_params_knn = grid_knn.best_params_
cv_results_knn = pd.DataFrame(grid_knn.cv_results_)
```

[44]:
```
print('Best Model:', best_model_knn)
print('Best Score:', round(best_score_knn,3))
print('Best Params:', best_params_knn)
print('CV Results')
display(display(cv_results_knn[['param_n_neighbors', 'param_weights',
 ↪'mean_test_score', 'std_test_score']]))
```

```
Best Model: KNeighborsClassifier(n_neighbors=3)
Best Score: nan
Best Params: {'n_neighbors': 3, 'weights': 'uniform'}
CV Results
```

	param_n_neighbors	param_weights	mean_test_score	std_test_score
0	3	uniform	NaN	NaN
1	3	distance	NaN	NaN
2	5	uniform	NaN	NaN
3	5	distance	NaN	NaN

```
None
```

2.0.8 Multinomial Logistic Regression

[45]:
```
from sklearn.linear_model import LogisticRegression
# Set up the Multinomial Logistic Regression model
mlr = LogisticRegression(multi_class='multinomial', solver='lbfgs',
 ↪random_state=42)
param_grid_mlr = {
    'C': [1, 10],
    'penalty': ['l2'],
    'max_iter': [1000, 1500]
```

```
}

# Perform Grid Search
grid_mlr = GridSearchCV(mlr, param_grid_mlr, cv=3, scoring='accuracy',␣
↪return_train_score=True)
grid_mlr.fit(X_train, y_train)
```

```
C:\Users\navee\anaconda3\Lib\site-
packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
C:\Users\navee\anaconda3\Lib\site-
packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
C:\Users\navee\anaconda3\Lib\site-
packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
C:\Users\navee\anaconda3\Lib\site-
packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

```
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[45]:
```
GridSearchCV(cv=3,
             estimator=LogisticRegression(multi_class='multinomial',
                                          random_state=42),
             param_grid={'C': [1, 10], 'max_iter': [1000, 1500],
                         'penalty': ['l2']},
             return_train_score=True, scoring='accuracy')
```

[46]:
```python
predictionsMLR = grid_mlr.predict(X_test)

# Evaluation
print(classification_report(y_test, predictionsMLR))
```

```
              precision    recall  f1-score   support

           0       0.84      0.98      0.91     57017
           1       0.00      0.00      0.00      1389
           2       0.54      0.14      0.22     10529

    accuracy                           0.83     68935
   macro avg       0.46      0.37      0.38     68935
weighted avg       0.78      0.83      0.78     68935


C:\Users\navee\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are 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, msg_start, len(result))
C:\Users\navee\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are 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, msg_start, len(result))
C:\Users\navee\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are 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, msg_start, len(result))
```

[47]:
```python
best_model_mlr = grid_mlr.best_estimator_
best_score_mlr = grid_mlr.best_score_
best_params_mlr = grid_mlr.best_params_
cv_results_mlr = pd.DataFrame(grid_mlr.cv_results_)
```

[48]:
```
print('Best Model:', best_model_mlr)
print('Best Score:', round(best_score_mlr,3))
print('Best Params:', best_params_mlr)
print('CV Results')
display(display(cv_results_mlr[['param_C', 'param_penalty', 'mean_test_score',␣
↪'std_test_score', 'mean_train_score', 'std_train_score']]))
```

```
Best Model: LogisticRegression(C=1, max_iter=1500, multi_class='multinomial',
                   random_state=42)
Best Score: 0.83
Best Params: {'C': 1, 'max_iter': 1500, 'penalty': 'l2'}
CV Results
```

```
  param_C param_penalty  mean_test_score  std_test_score  mean_train_score  \
0       1            l2         0.830440        0.000239          0.830478
1       1            l2         0.830459        0.000254          0.830490
2      10            l2         0.830440        0.000286          0.830499
3      10            l2         0.830434        0.000222          0.830499

   std_train_score
0         0.000040
1         0.000043
2         0.000050
3         0.000056
```

```
None
```

2.0.9 Support Vector Machine (SVM)

[ ]:
```
from sklearn.svm import SVC

#loading the classification
svc = SVC(random_state=42)

# Using grid search to tune for the best parameters
parameters_grid_svm = {
    'C': [1, 10],  # Tuning best regularization parameter
    'kernel': ['rbf', 'poly']  # Kernel to be used. Using kernels as they might␣
↪capture non linear relation if it exists
}

# Performing grid search and cross validating it 3 times.
grid_search_svm = GridSearchCV(svc, parameters_grid_svm, cv=3,␣
↪return_train_score=True)

#Training the model
grid_search_svm.fit(X_train, y_train)
```

```
[ ]:
```

2.0.10 Random Forest Classifier

Best Model : RandomForestClassifier(max_depth=20, n_estimators=200, random_state=42)

Best Score : 0.868

The classification report provides key metrics for each class (0, 1, 2) in the model:

Class 0: Shows high precision, recall, and F1-score, indicating that the model performs well in identifying and predicting non-diabetic outcomes. Class 1: Has extremely low precision, recall, and F1-score, suggesting that the model struggles significantly with predicting pre-diabetic outcomes accurately. Class 2: Has moderate precision and recall, but these metrics are not as high as for Class 0, indicating some challenges in predicting diabetic outcomes but still reasonably effective compared to Class 1. The model is biased towards predicting non-diabetic outcomes (Class 0) much more effectively than pre-diabetic or diabetic outcomes. The very poor performance on Class 1 could be due to underrepresentation in the dataset, or the features not being indicative enough for this class, or both.

Lower max_depth can lead to higher bias as the model is potentially too simple to capture the underlying patterns in the data. THe results suggest that a max_depth of 10 might be too simplistic as performance improves significantly with max_depth of 20. Higher max_depth can increase model complexity, which might lead to overfitting. However, the standard deviation of test scores (indicative of variability in model performance across different folds) does not increase substantially with higher max_depth, suggesting that the model does not suffer excessively from high variance, at least within the range tested.

The Random Forest model demonstrated robust performance with strong precision in predicting non-diabetic cases but struggled with lower accuracy in predicting pre-diabetic and diabetic cases.

2.0.11 KNN Classifier

Best Model: KNeighborsClassifier(n_neighbors=3, weights='distance')

Best Score: 0.877

The classification report provides the following metrics for each class:

Class 0 (Non-diabetic):

Precision: High at 0.89, indicating that the model is very good at not labeling non-diabetic cases as diabetic or pre-diabetic. Recall: Also relatively high at 0.80, showing that it successfully identifies a good portion of actual non-diabetic cases. F1-Score: An F1-score of 0.84 reflects a good balance between precision and recall for this class.

Class 1 (Pre-diabetic):

Precision: Very low at 0.03, suggesting the model rarely identifies pre-diabetic cases correctly. Recall: Also very low at 0.07, meaning it misses many actual pre-diabetic cases. F1-Score: At 0.04, the F1-score is extremely low, indicating poor performance for this class.

Class 2 (Diabetic):

Precision: Moderate at 0.29, indicating the model has some capability to correctly identify diabetic cases but also makes several false positives. Recall: At 0.41, it captures less than half of the actual diabetic cases. F1-Score: The score of 0.34 suggests modest effectiveness for this class but indicates room for improvement. Overall Accuracy: The overall accuracy is 0.73, which might seem reasonable but is heavily biased towards the majority class (Class 0).

There is a clear indication of bias towards the majority class (Class 0). The model performs well in identifying non-diabetic cases but struggles significantly with pre-diabetic cases, suggesting underfitting for minority classes. The relatively low standard deviation in test scores between different configurations suggests that the model isn't experiencing high variance.

The KNN classifier showed good performance for the non-diabetic class but significantly underperformed for pre-diabetic and diabetic classes, indicating difficulties in handling class imbalances.

2.0.12 Multinomial Logistic Regression

Best Model:LogisticRegression(C=1, max_iter=1000, multi_class='multinomial', random_state=42)

Best Score: 0.522

Class 0 (Non-diabetic):

Precision: High (0.95), indicating a strong accuracy in predicting non-diabetic outcomes when the model does predict this class. Recall: Moderate (0.64), suggesting that the model misses a significant number of true non-diabetic cases. F1-Score: Fairly high (0.76), showing a decent balance between precision and recall for this class.

Class 1 (Pre-diabetic):

Precision: Extremely low (0.03), meaning the model rarely predicts this class correctly. Recall: Low (0.30), indicating the model misses many pre-diabetic cases. F1-Score: Very poor (0.05), reflecting the model's ineffective performance for this class.

Class 2 (Diabetic):

Precision: Low (0.34), suggesting many false positives (non-diabetic or pre-diabetic cases wrongly identified as diabetic). Recall: Moderate (0.57), meaning the model captures over half of the actual diabetic cases but still misses a considerable number. F1-Score: Moderate (0.42), indicating a need for improvement in balancing precision and recall for this class.

The model shows a high bias towards Class 0, which it predicts quite well compared to the other classes. This could be indicative of a model that isn't complex enough to capture the nuances of Classes 1 and 2, possibly due to a lack of relevant features or insufficient representation of these classes in the training data. The model has low variance, as indicated by the small standard deviations in cross-validation scores, suggesting that changes in the training dataset have little effect on the outcome. This stability, while generally positive, comes at the cost of high bias — particularly toward predicting Classes 1 and 2 inadequately.

While the model performs consistently, it does so at a generally low level of accuracy, with significant room for improvement in its application for predicting diabetic outcomes. Addressing its high bias while maintaining low variance is crucial for enhancing its effectiveness in a healthcare setting. The

MLR model consistently underperformed across all classes, particularly struggling with pre-diabetic classifications, indicating a high bias toward the majority class.

Since this is a classification problem, the metrics used are accuracy, precision, recall and f1-score. In the specific case of healthcare, recall is preferred over other metrics. Recall measures the proportion of actual positives that are correctly identified. In many healthcare scenarios, especially those involving the diagnosis of serious conditions, a high recall is crucial because the cost of missing a true positive (failing to identify a disease) can be very high, potentially life-threatening. Additionally, even in cases where the disease prediction might be incorrect, a physician's assessment will ultimately confirm the presence or absence of diabetes, providing a secondary check.

Observation: Patients, especially those at risk of diabetes, stand to benefit directly as they can receive earlier and more targeted interventions, potentially preventing the progression of the disease. Healthcare providers, including clinicians and hospitals, will benefit from more accurate diagnostic tools, aiding in better patient management and optimized treatment plans. Additionally, healthcare systems will experience improved resource allocation and reduced costs by preventing severe complications through early treatment.

The models showed a tendency to perform well in predicting the majority class (non-diabetic) but struggled significantly with the minority classes (pre-diabetic and diabetic). This imbalance can lead to a model that is biased towards the majority class, potentially neglecting or misclassifying the more critical minority cases which require accurate identification for effective treatment and intervention.

[ ]:
```
plt.figure(figsize=(10, 6))
sns.countplot(x='Diabetes_012', hue='Sex', data=df)
plt.title('Distribution of Diabetic Outcomes by Sex')
plt.xlabel('Diabetic Status')
plt.ylabel('Count')
plt.legend(title='Sex', labels=['Female', 'Male'])
plt.show()
```

From the above plot, we can see that the number of females is slightly more than males. While there is a notable difference in the counts, the disproportion does not appear to be extreme. However, this imbalance could still introduce a slight bias in the model's ability to learn and predict diabetic outcomes equitably for both sexes.

[ ]:
```
plt.figure(figsize=(10, 6))
sns.countplot(x='Diabetes_012', hue='Age', data=data)
plt.title('Distribution of Diabetic Outcomes by Age Group')
plt.xlabel('Diabetic Status')
plt.ylabel('Count')
plt.legend(title='Age Group')
plt.show()
```

The second chart displays a clear trend of diabetic status across various age groups, with higher ages showing a significant increase in diabetic (2) outcomes. The representation across age groups is more balanced in the non-diabetic (0) status but shows variability in the pre-diabetic (1) and diabetic (2) statuses. This variability might suggest a bias where the model becomes better at predicting outcomes for age groups that are more heavily represented in the non-diabetic category, potentially skewing predictions for younger or older age groups that deviate from the majority. While there is a certain degree of imbalance in the dataset with respect to both sex and age group distributions, the bias introduced by these factors may not be substantial but should not be overlooked.

Adjusting class weights, SMOTE are some common techniques to address the class imbalance issue.

```
[ ]:
```