

Problem Statement

Objective

The goal of this project is to create a model that can predict the presence of diabetes in individuals by analyzing data from the Behavioral Risk Factor Surveillance System (BRFSS) 2015 survey. The main objective is to accurately forecast whether prediabetes (*Outcome* = 1) or no diabetes (*Outcome* = 0) based on specific health measurements and indicators.

Background

Diabetes poses a significant challenge to public health, impacting numerous Americans and leading to severe health complications and considerable financial strains. Timely identification and intervention can greatly enhance patient outcomes and appropriate treatments.

This initiative intends to harness machine learning methods to bolster the early recognition of diabetes, potentially assisting healthcare professionals in making well informed choices and enhancing patient well being.

Health Indicators and Indicators for Diagnosis

The model will consider a range of health indicators, such as high blood pressure, elevated cholesterol levels, recent cholesterol screenings, BMI, smoking habits, history of stroke or heart disease/attack, level of physical activity, consumption or access to healthcare services, financial obstacles hindering doctor visits, overall health condition perception, number of days with mental or physical health issues, mobility difficulties, gender identity, age groupings education attainment levels and more.

These signs offer a detailed overview of a person's well being and habits, which are essential for predicting the likelihood of developing diabetes.

Data Overview

The information utilized in this study is a refined and organized version of the BRFSS 2015 survey data found on Kaggle. It encompasses responses from more than 253,680 individuals and incorporates various health related measurements. This dataset indicates whether an individual does not have diabetes (0), has prediabetes (1) or has diabetes (2).

The dataset named "diabetes_012_health_indicators_BRFSS2015.csv" was selected for specific reasons:

- **Diverse Target Variable:** This dataset offers a three class target variable (no diabetes, prediabetes and diabetes), enabling a thorough examination and forecasting of different diabetes stages.
- **Substantial Sample Size:** With 253,680 survey responses, this dataset presents a significant sample size that boosts the trustworthiness and applicability of the predictive model.
- **Extensive Health Indicators:** Featuring 21 feature variables encompassing various health indicators, this dataset establishes a comprehensive foundation for predictive analysis.
- **Natural Class Imbalance:** In contrast to the "diabetes_binary_5050split_health_indicators_BRFSS2015.csv" dataset with an even split, this dataset mirrors the typical class imbalance observed in real world settings. This characteristic supports performance assessment.

Research Inquiries

This project seeks to investigate several primary research queries:

1. Can survey data effectively forecast an individual's diabetes status?
2. What are the most influential risk factors for diabetes prediction?
3. Can a subset of the characteristics accurately predict the likelihood of developing diabetes?
4. Is it possible to develop a concise set of survey questions for efficient screening of diabetes risk?

Significance

Developing a dependable model for diagnosing diabetes can significantly impact public health. It can aid in identifying high risk individuals, enabling prompt intervention and ultimately leading to improved disease management. This endeavor not only aims to predict diabetes risk but also aims to comprehend the fundamental elements contributing to diabetes risk, offering valuable insights for healthcare professionals and policymakers.

```
In [6]: # pip install -r Requirements.txt
In [8]: pip install dataprep --quiet
^C
Note: you may need to restart the kernel to use updated packages.

In [72]: import random
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score, cross_validate
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.decomposition import PCA
from sklearn.utils.class_weight import compute_class_weight
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB, GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report, mean_squared_error, r2_score, f1_score, accuracy_score, precision_score, recall_score, confusion_matrix, ConfusionMatrixDisplay, roc_auc_score, roc_curve, auc, precision_recall_curve
from xgboost import XGBClassifier
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.combine import SMOTETENN
from yellowbrick.classifier import ROCAUC
from scipy.stats import chi2_contingency, test_ind
import optuna

# suppress warnings
warnings.filterwarnings("ignore")

# from dataprep.eda import *
# from dataprep.datasets import load_dataset
# from dataprep.eda import plot, plot_correlation, plot_missing, plot_diff, create_report #

# Apply the seaborn theme
sns.set_theme() #overwrite default Matplotlib styling parameters
```

Data understanding (EDA)

```
In [10]: # Load data and print dataframe shape
file_path = "diabetes_012_health_indicators_BRFSS2015.csv"
df = pd.read_csv(file_path)

shape = df.shape
print("Shape of the dataframe (row, col):", shape, "\n")

# Show the dataframe
pd.set_option('display.max_columns', None)
df
```

Shape of the dataframe (row, col): (253680, 22)

Out[10]	Diabetes_012	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttack	PhysActivity	Fruits	Veggies	HvyAlcoholConsump	AnyHealthcare	NoDocbcCost	GenHlth	MentHlth	PhysHlth	DiffWalk	Sex	Age	Education		
0	0.0	1.0	1.0	1.0	40.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	5.0	18.0	15.0	1.0	0.0	9.0	4.0		
1	0.0	0.0	0.0	0.0	25.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	3.0	0.0	0.0	0.0	0.0	7.0	6.0		
2	0.0	1.0	1.0	1.0	28.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0	5.0	30.0	30.0	1.0	0.0	9.0	4.0		
3	0.0	1.0	0.0	1.0	27.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	2.0	0.0	0.0	0.0	11.0	3.0		
4	0.0	1.0	1.0	1.0	24.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	2.0	3.0	0.0	0.0	11.0	5.0		
...		
253675	0.0	1.0	1.0	1.0	45.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0	3.0	0.0	5.0	0.0	1.0	5.0	6.0	
253676	2.0	1.0	1.0	1.0	18.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	4.0	0.0	0.0	1.0	0.0	11.0	2.0	
253677	0.0	0.0	0.0	1.0	28.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	2.0	5.0	5.0	
253678	0.0	1.0	0.0	1.0	23.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0	3.0	0.0	0.0	0.0	1.0	7.0	5.0
253679	2.0	1.0	1.0	1.0	25.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	2.0	0.0	0.0	0.0	0.0	9.0	6.0	

253680 rows × 22 columns

In [1]: `display(df.info())`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 253680 entries, 0 to 253679
Data columns (total 22 columns):
 #   Column      Non-Null Count  Dtype  
 --- 
 0   Diabetes_012    253680 non-null float64
 1   HighBP       253680 non-null float64
 2   HighChol     253680 non-null float64
 3   CholCheck    253680 non-null float64
 4   BMI          253680 non-null float64
 5   Smoker       253680 non-null float64
 6   Stroke        253680 non-null float64
 7   HeartDiseaseorAttack  253680 non-null float64
 8   PhysActivity  253680 non-null float64
 9   Fruits        253680 non-null float64
 10  Veggies       253680 non-null float64
 11  HvyAlcoholConsump  253680 non-null float64
 12  AnyHealthcare  253680 non-null float64
 13  NoDocbcCost   253680 non-null float64
 14  GenHlth       253680 non-null float64
 15  MentHlth      253680 non-null float64
 16  PhysHlth      253680 non-null float64
 17  DiffWalk      253680 non-null float64
 18  Sex           253680 non-null float64
 19  Age           253680 non-null float64
 20  Education     253680 non-null float64
 21  Income         253680 non-null float64
dtypes: float64(22)
memory usage: 42.6 MB
None
```

Detailed Variable Descriptions

1. **Diabetes_012** :
 - Categorical: 0 = No diabetes, 1 = Prediabetes, 2 = Diabetes.
2. **HighBP** :
 - Binary: 0 = No high blood pressure, 1 = High blood pressure.
3. **HighChol** :
 - Binary: 0 = No high cholesterol, 1 = High cholesterol.
4. **CholCheck** :
 - Binary: 0 = No cholesterol check in the last 5 years, 1 = Cholesterol check in the last 5 years.
5. **BMI** (Body Mass Index):
 - Continuous: A key indicator of body fat based on height and weight.
6. **Smoker** :
 - Binary: 0 = Non-smoker, 1 = Smoker (100+ cigarettes in life).
7. **Stroke** :
 - Binary: 0 = No stroke, 1 = Stroke.
8. **HeartDiseaseorAttack** :
 - Binary: 0 = No heart disease or attack, 1 = Heart disease or attack.
9. **PhysActivity** :
 - Binary: 0 = No physical activity, 1 = Physical activity.
10. **Fruits** :
 - Binary: 0 = Less than once per day, 1 = One or more times per day.
11. **Veggies** :
 - Binary: 0 = Less than once per day, 1 = One or more times per day.
12. **HvyAlcoholConsump** :
 - Binary: 0 = Not a heavy drinker, 1 = Heavy drinker (adult men >14 drinks/week, women >7 drinks/week).
13. **AnyHealthcare** :
 - Binary: 0 = No healthcare coverage, 1 = Healthcare coverage.
14. **NoDocbcCost** :
 - Binary: 0 = No cost barriers to healthcare, 1 = Cost barriers to healthcare.
15. **GenHlth** :
 - Ordinal (1-5): 1 = Excellent, 5 = Poor general health.
16. **MentHlth** :
 - Continuous (0-30 days): Number of days with poor mental health.
17. **PhysHlth** :
 - Continuous (0-30 days): Number of days with poor physical health.
18. **DiffWalk** :
 - Binary: 0 = No difficulty walking/climbing stairs, 1 = Difficulty.
19. **Sex** :
 - Binary: 0 = Female, 1 = Male.
20. **Age** :
 - Categorical (1-13 age groups): 1 = 18-24 years, 9 = 60-64 years, 13 = 80 or older.
21. **Education** :
 - Ordinal (1-6): 1 = No schooling, 6 = College graduate.
22. **Income** :
 - Ordinal (1-8): 1 = Less than 10,000, 8 = 75,000 or more.

```
In [1]: print(df.describe())
```

```
Diabetes_012    HighBP    HighChol    CholCheck \
count 253680.000000 253680.000000 253680.000000 253680.000000
mean 0.296921 0.429001 0.424211 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    Veggies HvyAlcoholConsump \
count 253680.000000 253680.000000 253680.000000 253680.000000
mean 0.756544 0.634256 0.811420 0.056197
std 0.429169 0.481639 0.391175 0.230302
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

  AnyHealthcare NoDocbcCost    GenHlth    MentHlth \
count 253680.000000 253680.000000 253680.000000 253680.000000
mean 0.951053 0.084177 2.511392 3.184772
std 0.215759 0.277654 1.068477 7.412847
min 0.000000 0.000000 1.000000 0.000000
25% 1.000000 0.000000 2.000000 0.000000
50% 1.000000 0.000000 2.000000 0.000000
75% 1.000000 0.000000 3.000000 2.000000
max 1.000000 1.000000 5.000000 30.000000

  PhysHlth    DiffWalk    Sex     Age \
count 253680.000000 253680.000000 253680.000000 253680.000000
mean 4.242081 0.168224 0.440342 8.032119
std 8.717951 0.374066 0.496429 3.054220
min 0.000000 0.000000 0.000000 1.000000
25% 0.000000 0.000000 0.000000 6.000000
50% 0.000000 0.000000 0.000000 8.000000
75% 3.000000 0.000000 1.000000 10.000000
max 30.000000 1.000000 1.000000 13.000000

  Education    Income \
count 253680.000000 253680.000000
mean 5.050434 6.053875
std 0.985774 2.071148
min 1.000000 1.000000
25% 4.000000 5.000000
50% 5.000000 7.000000
75% 6.000000 8.000000
max 6.000000 8.000000
```

```
In [2]: print(df.shape)
```

```
(253680, 22)
```

Initial Plot for EDA

```
In [3]: plot(df)
```

```
Out[3]:
```

Show Stats and Insights

Number of plots per page: -

Diabetes_012		HighBP		HighChol	
CholCheck		BMI		Smoker	
Stroke		HeartDiseaseorAttack		PhysActivity	
Fruits		Veggies		HvyAlcoholConsump	
AnyHealthcare		NoDocbcCost		GenHlth	
MentHlth		PhysHlth		DiffWalk	
Sex		Age		Education	
Income					

Dataset Overview

Total Variables: 22 Total Rows: 253,680 Missing Data: None reported Duplicate Rows: 23,899 (9.4%) Memory Usage: 42.6 MB Data Types: All variables are numeric but largely categorical by nature.

Variable Descriptions

Categorical (Binary or Multi-Class): Most of the variables are categorical, with binary encoding (e.g., HighBP, Smoker, PhysActivity) except Diabetes_012, which is a multi-class categorical variable. Numerical: BMI is a true continuous variable ↗ category. Key Insights and Considerations No Missing Values: This indicates that the dataset is complete, which simplifies preprocessing but should be verified for correctness (e.g., no improper imputation).

Duplicates: The presence of about 9.4% duplicate rows should be addressed. Determine if these duplicates are due to data entry errors or if they represent legitimate repeated measurements.

Variable Characteristics:

Diabetes Status (Diabetes_012) : Captures the absence or presence of diabetes and its stage, useful for detailed health-related analyses. Lifestyle and Health Checks (Smoker, CholCheck) : Reflect lifestyle choices and adherence to health metrics (BMI, PhysActivity) : Directly measures aspects of physical health and activity, crucial for studying correlations with health outcomes.

Data Skewness and Distribution:

Skewed Variables (MentHlth, PhysHlth) : High zero count suggests many participants report no issues, which might require special modeling techniques like zero-inflated models if predicting these conditions. BMI Distribution: Given its role as (e.g., normality, presence of outliers) is important for understanding population health metrics. Binary Variables: Many variables are binary, indicating conditions like high blood pressure, smoking status, or having had a stroke. These will be part of statistical tests to determine risk factors for various health conditions.

Understanding Correlation

1. Correlation Overview

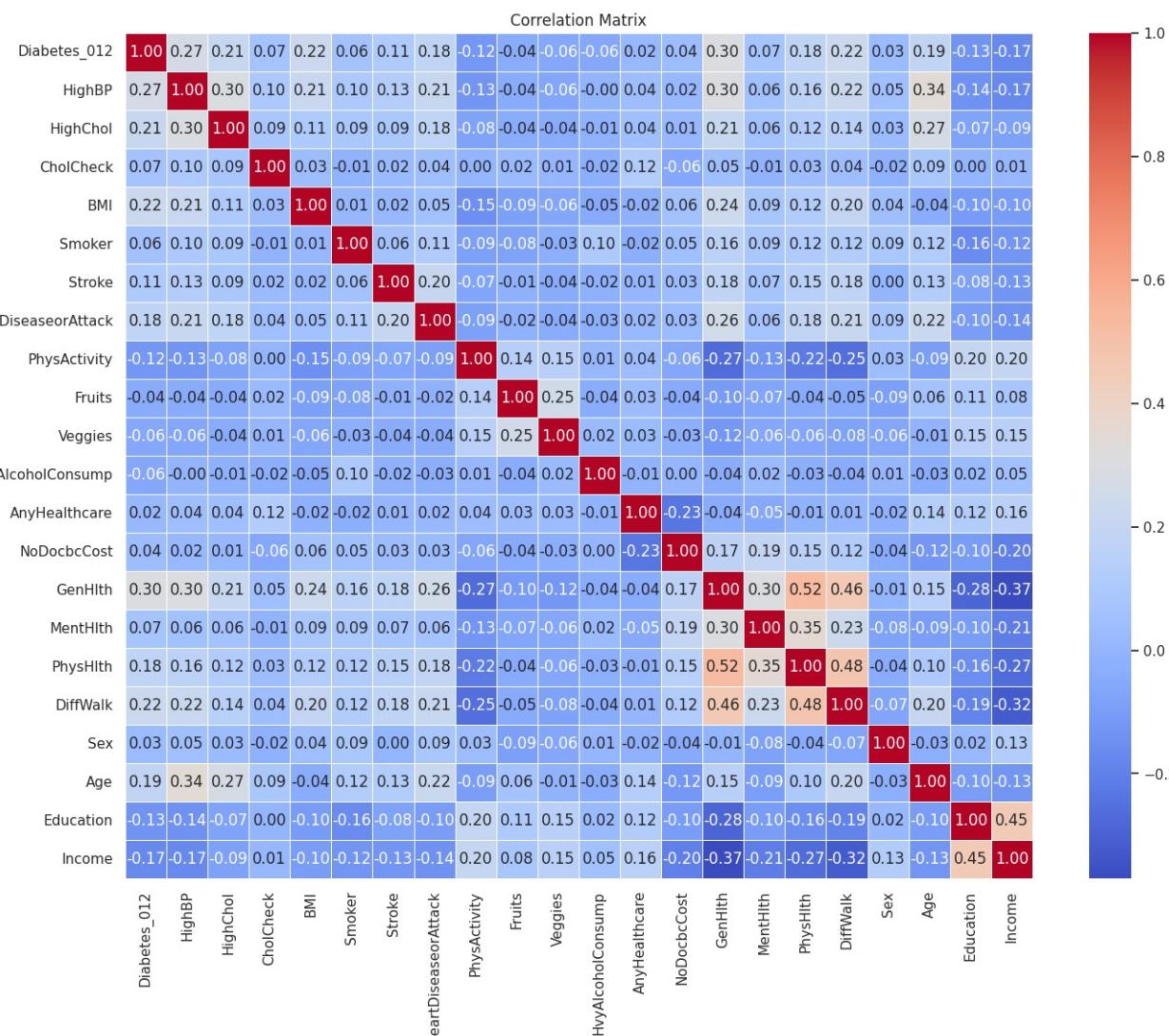
```
In [4]: plot_correlation(df)
```

Out[]:	Stats	Pearson	Spearman	KendallTau
		Pearson	Spearman	KendallTau
	Highest Positive Correlation	0.524	0.452	0.39
	Highest Negative Correlation	-0.37	-0.354	-0.292
	Lowest Correlation	0.002	0.003	0.003
	Mean Correlation	0.032	0.03	0.028

In []: correlation = df.corr()
correlation

Out[]:	Diabetes_012	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttack	PhysActivity	Fruits	Veggies	HvyAlcoholConsump	AnyHealthcare	NoDocbcCost	GenHlth	MentHlth	PhysHlth	
	Diabetes_012	1.000000	0.271596	0.209085	0.067546	0.224379	0.062914	0.107179	0.180272	-0.121947	-0.042192	-0.058972	-0.057882	0.015410	0.035436	0.302587	0.073507	0.176287
	HighBP	0.271596	1.000000	0.298199	0.098508	0.213748	0.096991	0.129575	0.209361	-0.125267	-0.040555	-0.061266	-0.003972	0.038425	0.017358	0.300530	0.056456	0.161212
	HighChol	0.209085	0.298199	1.000000	0.085642	0.106722	0.091299	0.092620	0.180765	-0.078046	-0.040859	-0.039874	-0.011543	0.042230	0.013310	0.208426	0.062069	0.121751
	CholCheck	0.067546	0.098508	0.085642	1.000000	0.034495	-0.009929	0.024158	0.044206	0.004190	0.023849	0.006121	-0.023730	0.117626	-0.058255	0.046589	-0.008366	0.031775
	BMI	0.224379	0.213748	0.106722	0.034495	1.000000	0.013804	0.020153	0.052904	-0.147294	-0.087518	-0.062275	-0.048736	-0.018471	0.058206	0.239185	0.085310	0.121141
	Smoker	0.062914	0.096991	0.091299	-0.009929	0.013804	1.000000	0.061173	0.114441	-0.087401	-0.077666	-0.030678	0.101619	-0.023251	0.048946	0.163143	0.092196	0.116460
	Stroke	0.107179	0.129575	0.092620	0.024158	0.020153	0.061173	1.000000	0.203002	-0.069151	-0.013389	-0.041124	-0.016950	0.008776	0.034804	0.177942	0.070172	0.148944
	HeartDiseaseorAttack	0.180272	0.209361	0.180765	0.044206	0.052904	0.114441	0.203002	1.000000	-0.087299	-0.019790	-0.039167	-0.028991	0.018734	0.031000	0.258383	0.064621	0.181698
	PhysActivity	-0.121947	-0.125267	-0.078046	0.004190	-0.147294	-0.087401	-0.069151	-0.087299	1.000000	0.142756	0.153150	0.012392	0.035505	-0.061638	-0.266186	-0.125587	-0.219230
	Fruits	-0.042192	-0.040555	-0.040859	0.023849	-0.087518	-0.077666	-0.013389	-0.019790	0.142756	1.000000	0.254342	-0.035288	0.031544	-0.044243	-0.103854	-0.068217	-0.044633
	Veggies	-0.058972	-0.061266	-0.039874	0.006121	-0.062275	-0.030678	-0.041124	-0.039167	0.153150	0.254342	1.000000	0.021064	0.029584	-0.032232	-0.123066	-0.058884	-0.064290
	HvyAlcoholConsump	-0.057882	-0.003972	-0.011543	-0.023730	-0.048736	0.101619	-0.016950	-0.028991	0.012392	-0.035288	0.021064	1.000000	-0.010488	0.004684	-0.036724	0.024716	-0.026415
	AnyHealthcare	0.015410	0.038425	0.042230	0.117626	-0.018471	-0.023251	0.008776	0.018734	0.035505	0.031544	0.029584	-0.010488	1.000000	-0.232532	-0.040817	-0.052707	-0.008276
	NoDocbcCost	0.035436	0.017358	0.013310	-0.058255	0.058206	0.048946	0.034804	0.031000	-0.061638	-0.044243	-0.032232	0.004684	-0.232532	1.000000	0.166397	0.192107	0.148998
	GenHlth	0.302587	0.300530	0.208426	0.046589	0.239185	0.163143	0.177942	0.258383	-0.266186	-0.103854	-0.123066	-0.036724	-0.040817	0.166397	1.000000	0.301674	0.524364
	MentHlth	0.073507	0.056456	0.062069	-0.008366	0.085310	0.092196	0.070172	0.064621	-0.125587	-0.068217	-0.058884	0.024716	-0.052707	0.192107	0.301674	1.000000	0.353619
	PhysHlth	0.176287	0.161212	0.121751	0.031775	0.121141	0.116460	0.148944	0.181698	-0.219230	-0.044633	-0.064290	-0.026415	-0.008276	0.148998	0.524364	0.353619	1.000000
	DiffWalk	0.224239	0.223618	0.144672	0.040585	0.197078	0.122463	0.176567	0.212709	-0.253174	-0.048352	-0.080506	-0.037668	0.007074	0.118447	0.456920	0.233688	0.478417
	Sex	0.031040	0.052207	0.031205	-0.022115	0.042950	0.093662	0.002978	0.086096	0.032482	-0.091175	-0.064765	0.005740	-0.019405	-0.044931	-0.060691	-0.080705	-0.043137
	Age	0.185026	0.344452	0.272318	0.090321	-0.036618	0.120641	0.126974	0.221618	-0.092511	0.064547	-0.009771	-0.034578	0.138046	-0.119777	0.152450	-0.092068	0.099130
	Education	-0.130517	-0.141358	-0.070802	0.001510	-0.103932	-0.161955	-0.076009	-0.099600	0.199658	0.110187	0.154329	0.023997	0.122514	-0.100701	-0.284912	-0.101830	-0.155093
	Income	-0.171483	-0.171235	-0.085459	0.014259	-0.100069	-0.123937	-0.128599	-0.141011	0.198539	0.079929	0.151087	0.053619	0.157999	-0.203182	-0.370014	-0.209806	-0.266799

```
# use sns heatmap to plot the correlation matrix
plt.figure(figsize=(16, 12))
sns.heatmap(correlation, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Correlation Matrix')
plt.show()
```



- **Diabetes_012** and **GenHlth** (0.302587): Higher diabetes categorization correlates with worse general health ratings. This suggests that as diabetes severity increases, overall health perceptions tend to decline.
- **HighBP** and **Age** (0.344452): Older age groups tend to have higher incidences of high blood pressure.
- **PhysHlth** and **GenHlth** (0.524364): More days with poor physical health correlate strongly with poorer general health ratings.
- **MentHlth** and **PhysHlth** (0.353619): A significant positive correlation indicating that more days with mental health issues are associated with more days of poor physical health.
- **DiffWalk** and **PhysHlth** (0.478417): Difficulty walking correlates with more days of poor physical health, suggesting mobility issues are associated with worse physical conditions.

Notable Negative Correlations

- **PhysActivity** and **GenHlth** (-0.266186): Higher levels of physical activity correlate with better general health ratings.
- **Income** and **GenHlth** (-0.370014): Higher income levels correlate with better general health, highlighting possible socioeconomic impacts on health.
- **Education** and **GenHlth** (-0.284912): Higher education levels are associated with better general health ratings.

Implications for Analysis

- **Health Outcomes:** Variables like **GenHlth**, **Diabetes_012**, **HighBP**, and **HighChol** can be used to model health outcomes, especially for understanding risk factors associated with chronic diseases.
- **Behavioral Factors:** The relationships between lifestyle choices (e.g., **Smoker**, **PhysActivity**, **Fruits**, **Veggies**) and health outcomes can inform public health interventions.
- **Socioeconomic Factors:** The strong correlations between **Income**, **Education**, and health variables suggest that socioeconomic factors are crucial determinants of health, which could be a focal point for deeper socioeconomic studies and interventions.

In [1]: %matplotlib inline

```
def plot_all_correlations_individual(dataframe):
    for column in dataframe.columns:
        # The function plot_correlation from dataprep to plot each column against all others
        plot_correlation(dataframe, column)
        print("Plotting correlations for: " + column)

# call the function with your DataFrame
plot_all_correlations_individual(df)
```

Plotting correlations for: Diabetes_012

Plotting correlations for: HighBP

Plotting correlations for: HighChol

Plotting correlations for: CholCheck

Plotting correlations for: BMI

Plotting correlations for: Smoker

Plotting correlations for: Stroke

Plotting correlations for: HeartDiseaseorAttack

Plotting correlations for: PhysActivity

Plotting correlations for: Fruits

Plotting correlations for: Veggies

Plotting correlations for: HvyAlcoholConsump

Plotting correlations for: AnyHealthcare

Plotting correlations for: NoDocbcCost

Plotting correlations for: GenHlth

Plotting correlations for: MentHlth

Plotting correlations for: PhysHlth

Plotting correlations for: DiffWalk

Plotting correlations for: Sex

Plotting correlations for: Age

Plotting correlations for: Education

Plotting correlations for: Income

In [1]: plot_correlation(df, "Diabetes_012")

Pearson	Spearman	KendallTau
---------	----------	------------

In [1]: plot_correlation(df, "HighBP")

Pearson	Spearman	KendallTau
---------	----------	------------

In [1]: plot_correlation(df, "HighChol")

Pearson	Spearman	KendallTau
---------	----------	------------

In [1]: plot_correlation(df, "CholCheck")

Pearson	Spearman	KendallTau
---------	----------	------------

In [1]: plot_correlation(df, "BMI")

Pearson	Spearman	KendallTau
---------	----------	------------

In [1]: plot_correlation(df, "Smoker")

Pearson	Spearman	KendallTau
---------	----------	------------

In [1]: plot_correlation(df, "Stroke")

Pearson	Spearman	KendallTau
---------	----------	------------

In [1]: plot_correlation(df, "HeartDiseaseorAttack")

Pearson	Spearman	KendallTau
---------	----------	------------

In [1]: plot_correlation(df, "PhysActivity")

Pearson	Spearman	KendallTau
---------	----------	------------

Data Preparation and Identifying and printing the numerical, categorical, and binary/flag features in the DataFrame for further EDA.

In [13]: # Classify variables

```
def classify_variables():
    numerical = ['BMI', 'MentHlth', 'PhysHlth']
    categorical = ['Diabetes_012', 'Age']
    binary = ['HighBP', 'HighChol', 'CholCheck', 'Smoker', 'Stroke', 'HeartDiseaseorAttack',
              'PhysActivity', 'Fruits', 'Veggies', 'HvyAlcoholConsump', 'AnyHealthcare',
              'NoDocbcCost', 'DiffWalk', 'Sex']
    ordinal = ['GenHlth', 'Education', 'Income']
    return numerical, categorical, binary, ordinal
```

```

numerical, categorical, binary, ordinal = classify_variables()
non_numerical = categorical + binary + ordinal # All non-numerical variables
print("Numerical Variables:", numerical)
print("Non Numerical Variables:", non_numerical)
print("Categorical Variables:", categorical)
print("Binary Variables:", binary)
print("Ordinal Variables:", ordinal)

Numerical Variables: ['BMI', 'MentHlth', 'PhysHlth']
Non Numerical Variables: ['Diabetes_012', 'Age', 'HighBP', 'HighChol', 'CholCheck', 'Smoker', 'Stroke', 'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies', 'HvyAlcoholConsump', 'AnyHealthcare', 'NoDocbcCost', 'DiffWalk', 'Sex', 'GenHlth', 'Education', 'Income']
Categorical Variables: ['Diabetes_012', 'Age']
Binary Variables: ['HighBP', 'HighChol', 'CholCheck', 'Smoker', 'Stroke', 'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies', 'HvyAlcoholConsump', 'AnyHealthcare', 'NoDocbcCost', 'DiffWalk', 'Sex']
Ordinal Variables: ['GenHlth', 'Education', 'Income']

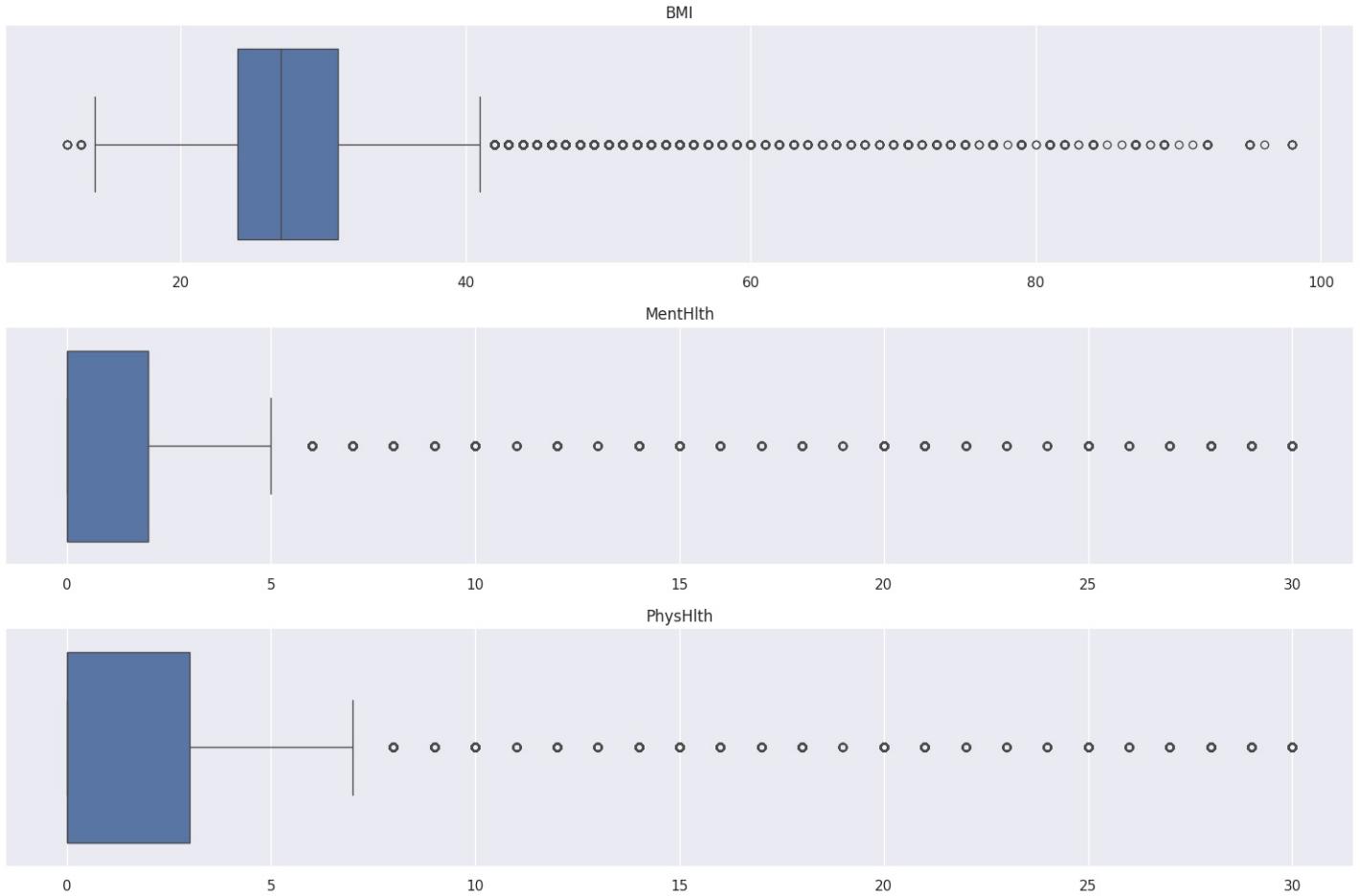
```

Univariate Analysis

```
# Set up the matplotlib figure and size
plt.figure(figsize=(15, 10))
```

```
# Iterate over each numerical feature to create a boxplot
for index, feature in enumerate(numerical):
    plt.subplot(len(numerical), 1, index + 1)
    sns.boxplot(x=df[feature])
    plt.title(feature)
    plt.xlabel("") # Remove x-labels as they are redundant with titles

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



Skewness: All three variables (BMI, MentHlth, PhysHlth) show skewness towards lower values, with the median close to the lower quartile.

Outliers: Each variable has significant outliers on the higher end, which could be important data points representing individuals with severe conditions.

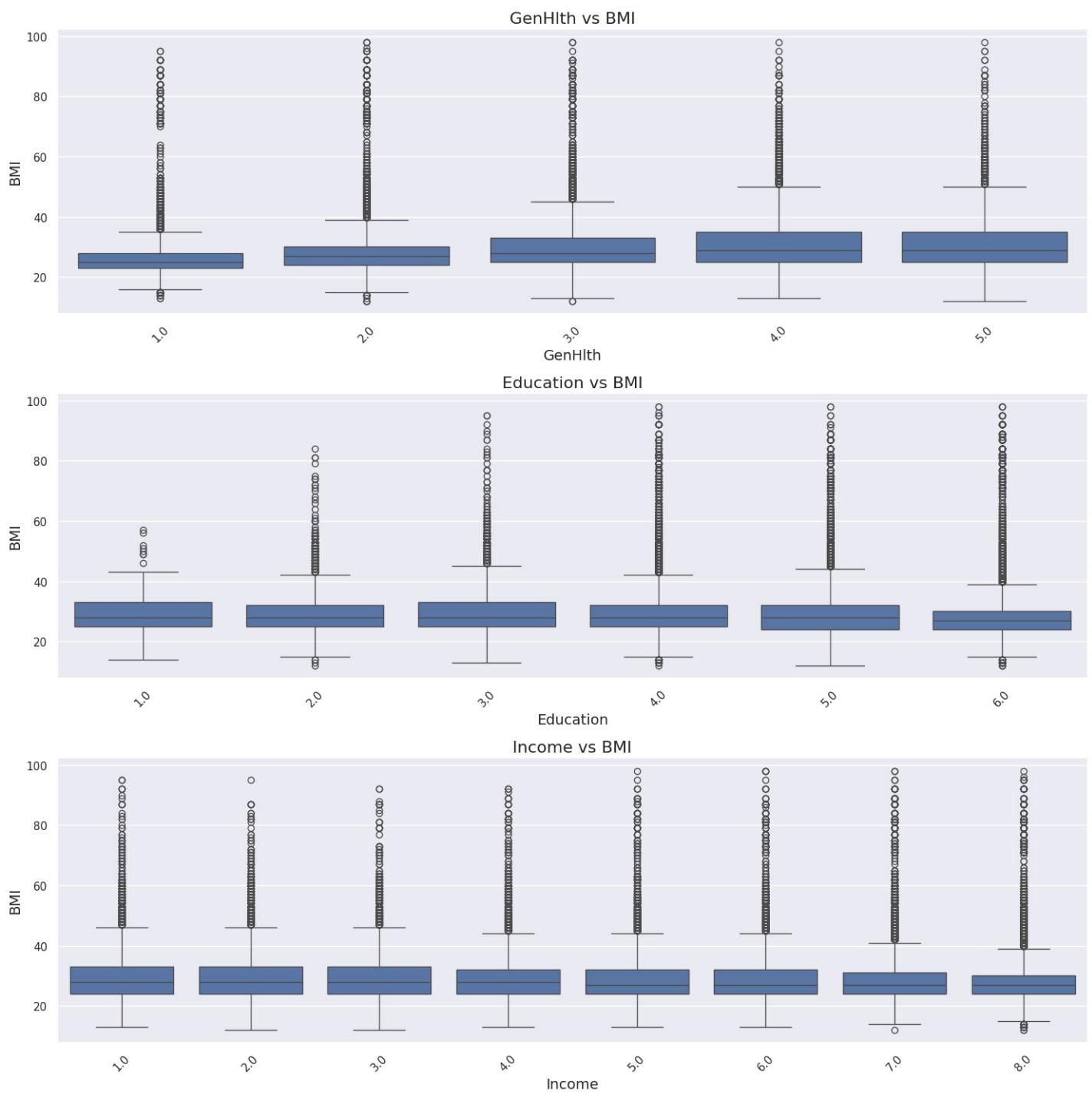
Central Tendency and Spread: The central tendency of all three variables is skewed towards lower values, and the IQR is relatively small, indicating that most data points are close to the median. However, the presence of outliers significantly affects the spread of the data.

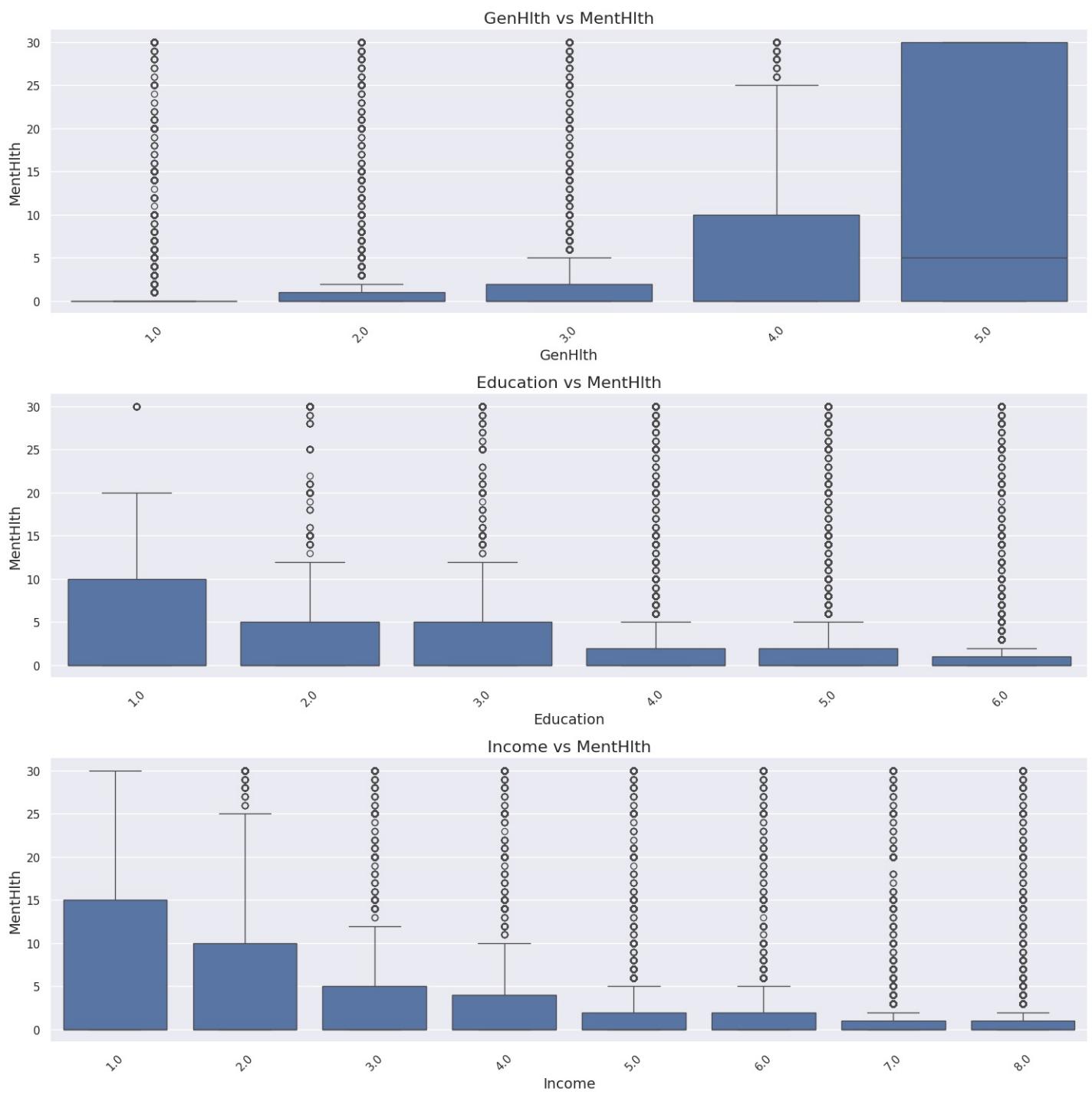
Univariate Analysis for Numerical vs Ordinal Variables

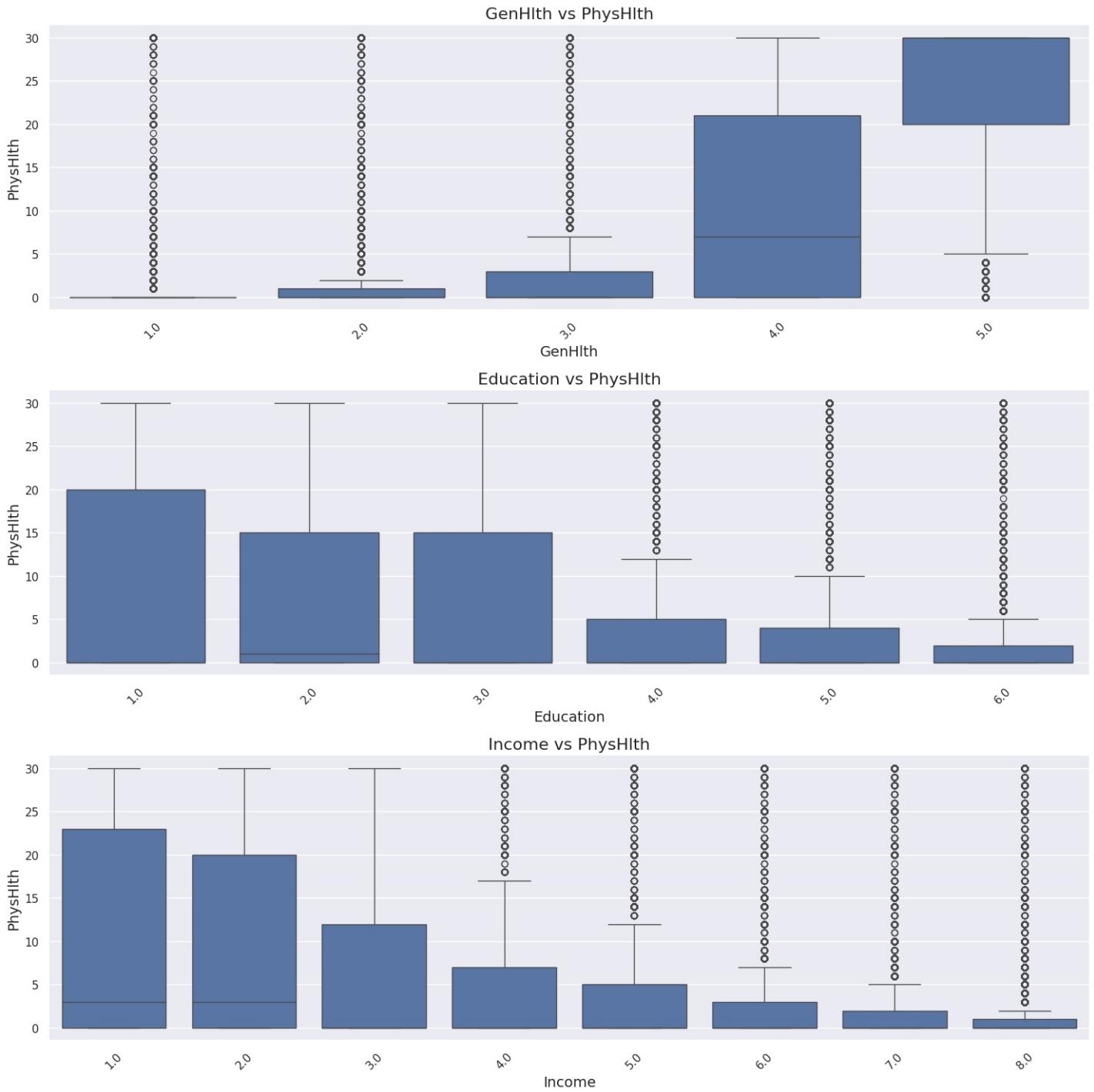
```
# Iterate over each numerical feature
for num_feature in numerical:
    # Set up the matplotlib figure and size
    plt.figure(figsize=(15, len(ordinal) * 5))
```

```
# Iterate over each ordinal feature to create a box plot
for index, ord_feature in enumerate(ordinal):
    plt.subplot(len(ordinal), 1, index + 1)
    sns.boxplot(data=df, x=ord_feature, y=num_feature)
    plt.title(f'{ord_feature} vs {num_feature}', fontsize=16)
    plt.xlabel(ord_feature, fontsize=14)
    plt.ylabel(num_feature, fontsize=14)
    plt.xticks(rotation=45) # Rotate x-axis labels by 45 degrees for readability

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







BMI Analysis

1. GenHlth vs BMI:

- This plot shows the relationship between general health (GenHlth) and Body Mass Index (BMI). There is a slight increase in BMI as general health worsens, but the median BMI values remain relatively consistent across different levels category.

2. Education vs BMI:

- This plot illustrates the relationship between education level and BMI. Higher education levels (4, 5, 6) show a slightly lower median BMI compared to lower education levels (1, 2, 3), suggesting that higher education might be associated with lower BMI.

3. Income vs BMI:

- This plot examines the relationship between income and BMI. There is a slight decrease in median BMI as income increases, indicating that higher income levels might be associated with lower BMI, though the difference is not pronounced.

Mental Health Analysis

1. GenHlth vs MentHlth:

- This plot shows the relationship between general health (GenHlth) and mental health (MentHlth). As general health worsens, the number of days with poor mental health increases significantly, especially for the poorest health category.

2. Education vs MentHlth:

- This plot illustrates the relationship between education level and mental health. Higher education levels (4, 5, 6) are associated with fewer days of poor mental health, indicating a possible link between higher education and better mental health.

3. Income vs MentHlth:

- This plot examines the relationship between income and mental health. Higher income levels are associated with fewer days of poor mental health, suggesting that higher income might contribute to better mental health.

Physical Health Analysis

1. GenHlth vs PhysHlth:

- This plot shows the relationship between general health (GenHlth) and physical health (PhysHlth). As general health worsens, the number of days with poor physical health increases dramatically, particularly in the poorest health category.

2. Education vs PhysHlth:

- This plot illustrates the relationship between education level and physical health. Higher education levels (4, 5, 6) are associated with fewer days of poor physical health, indicating that higher education might be linked to better physical health.

3. Income vs PhysHlth:

- This plot examines the relationship between income and physical health. Higher income levels are associated with fewer days of poor physical health, suggesting that higher income might contribute to better physical health.

Univariate Analysis for Numerical vs Categorical Variables

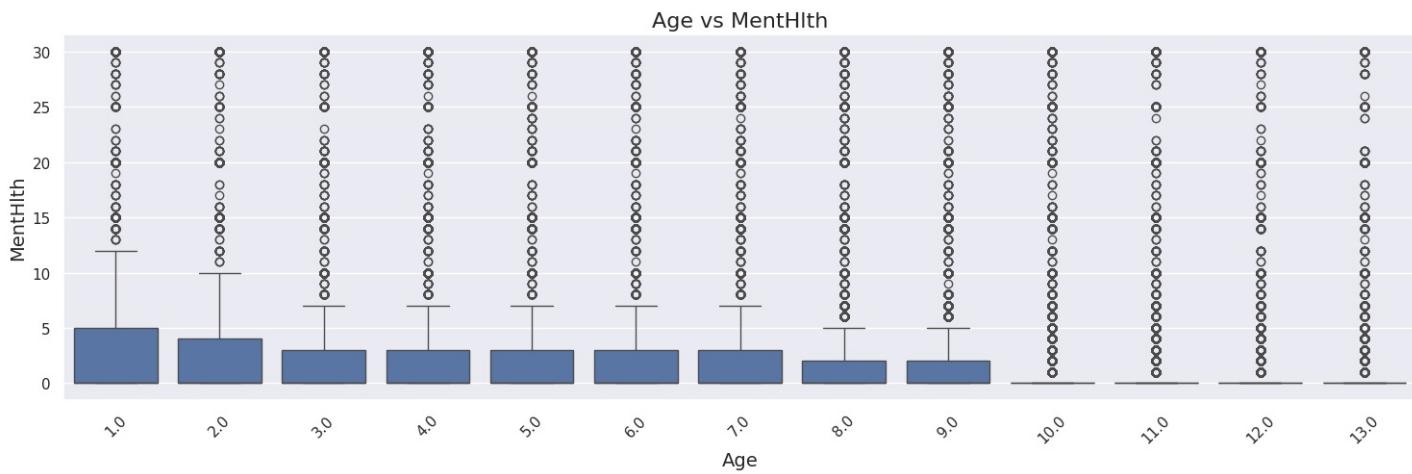
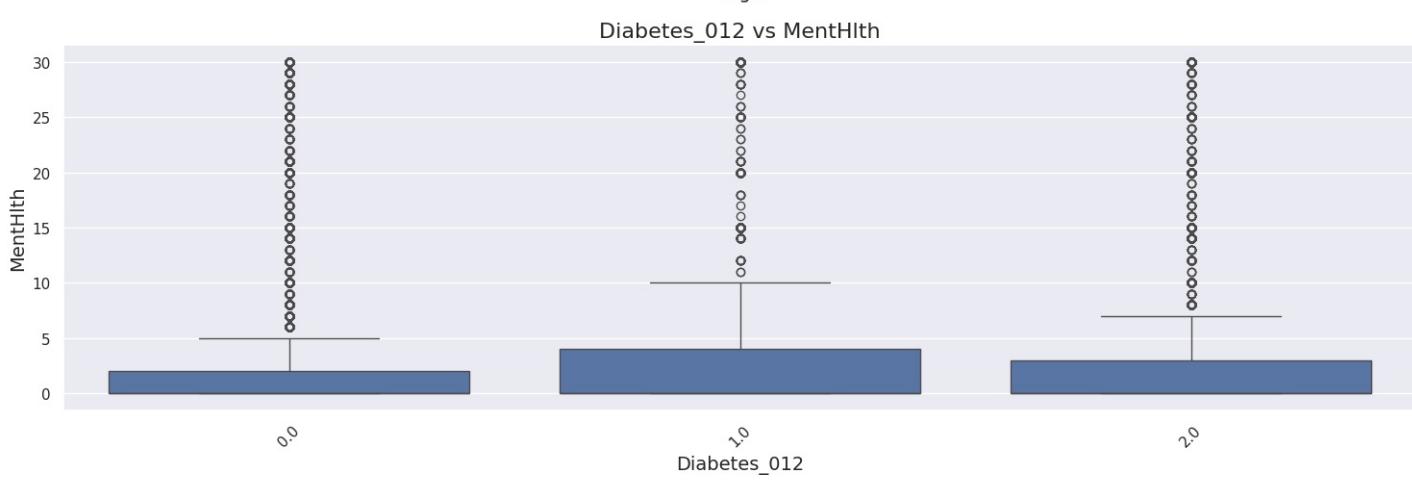
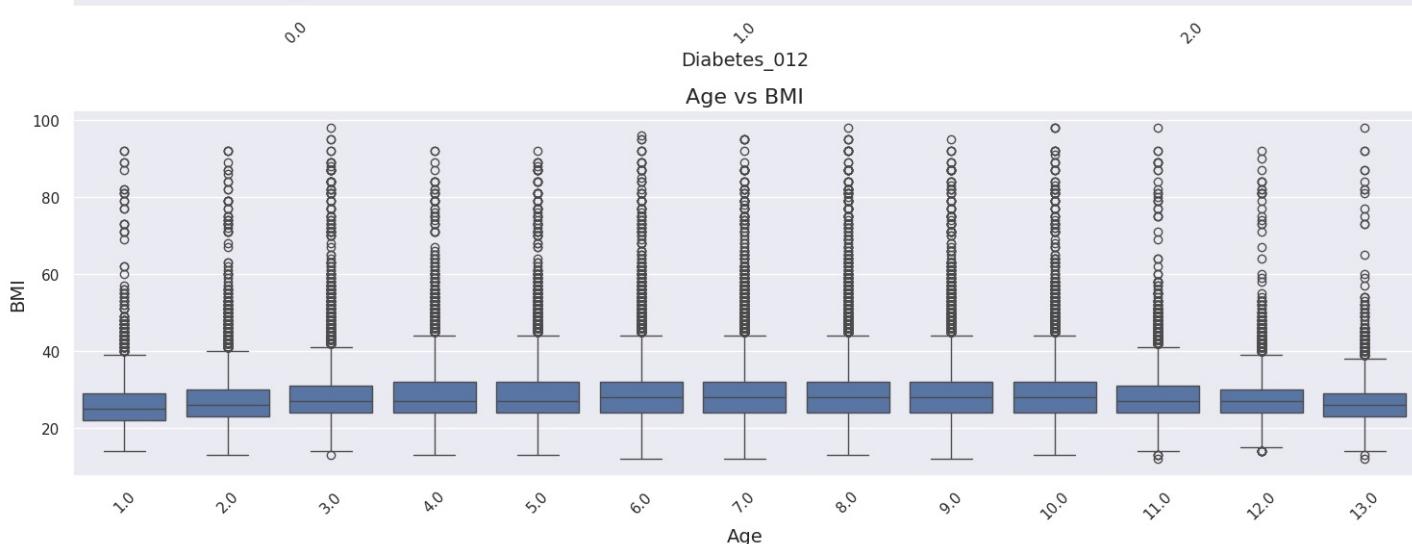
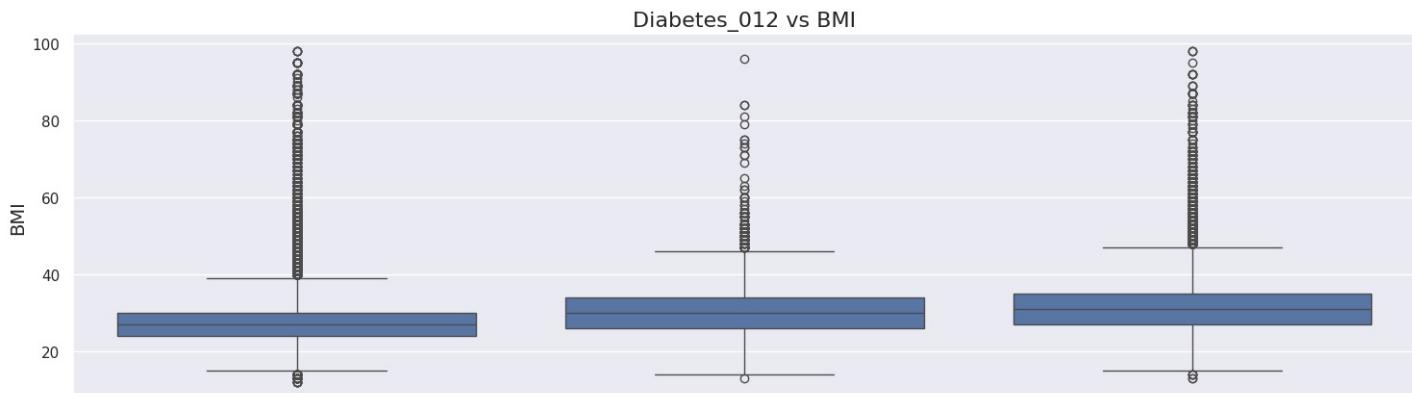
```

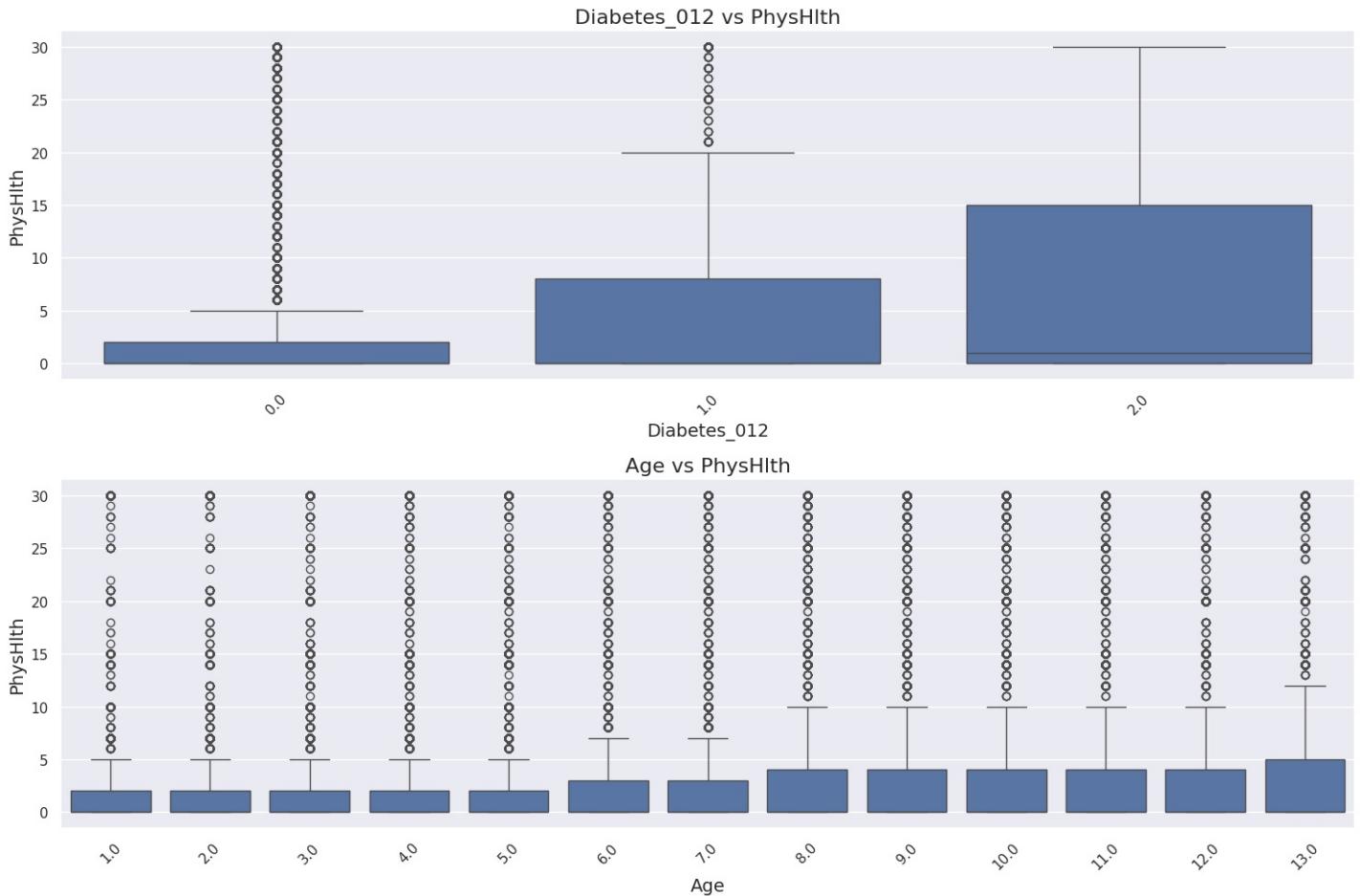
In [1]: # Iterate over each numerical feature
for num_feature in numerical:
    # Set up the matplotlib figure and size
    plt.figure(figsize=(15, len(categorical) * 5))

    # Iterate over each categorical feature to create a box plot
    for index, ord_feature in enumerate(categorical):
        plt.subplot(len(categorical), 1, index + 1)
        sns.boxplot(data=df, x=ord_feature, y=num_feature)
        plt.title(f'{ord_feature} vs {num_feature}', fontsize=16)
        plt.xlabel(ord_feature, fontsize=14)
        plt.ylabel(num_feature, fontsize=14)
        plt.xticks(rotation=45) # Rotate x-axis labels by 45 degrees for readability

    plt.tight_layout()
    plt.show()

```





BMI Analysis

1. Diabetes_012 vs BMI:

- This plot shows the relationship between diabetes status (Diabetes_012) and Body Mass Index (BMI). There is an increasing trend in median BMI from those without diabetes (0) to prediabetes (1) and diabetes (2). Individuals with diabetes show a higher median and upper quartile values.

2. Age vs BMI:

- This plot illustrates the relationship between age groups and BMI. BMI appears relatively stable across different age groups, with a slight decrease in the upper quartile for older age groups (10-13). There are numerous outliers in each group.

Mental Health Analysis

1. Diabetes_012 vs MentHlth:

- This plot shows the relationship between diabetes status and mental health (MentHlth). Individuals with diabetes (2) have a higher median number of poor mental health days compared to those without diabetes (0) and prediabetes (1).

2. Age vs MentHlth:

- This plot examines the relationship between age groups and mental health. The median number of poor mental health days is relatively consistent across age groups, with a slight decrease in older age groups (10-13). Outliers are present in each group.

Physical Health Analysis

1. Diabetes_012 vs PhysHlth:

- This plot shows the relationship between diabetes status and physical health (PhysHlth). Individuals with diabetes (2) have a higher median number of poor physical health days compared to those without diabetes (0) and prediabetes (1).

2. Age vs PhysHlth:

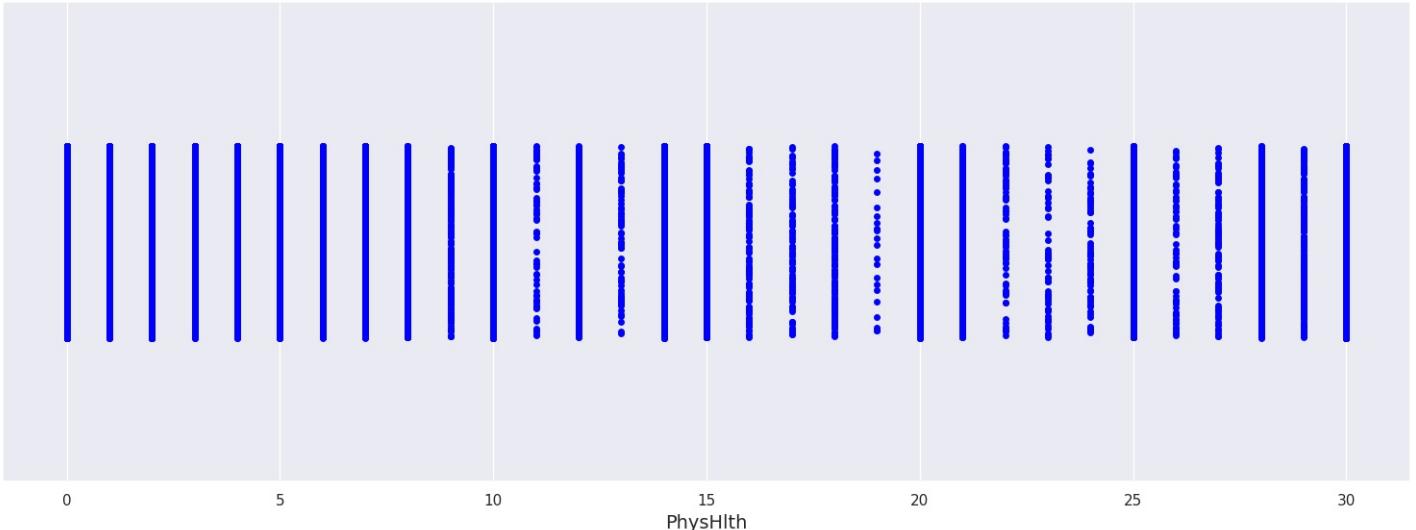
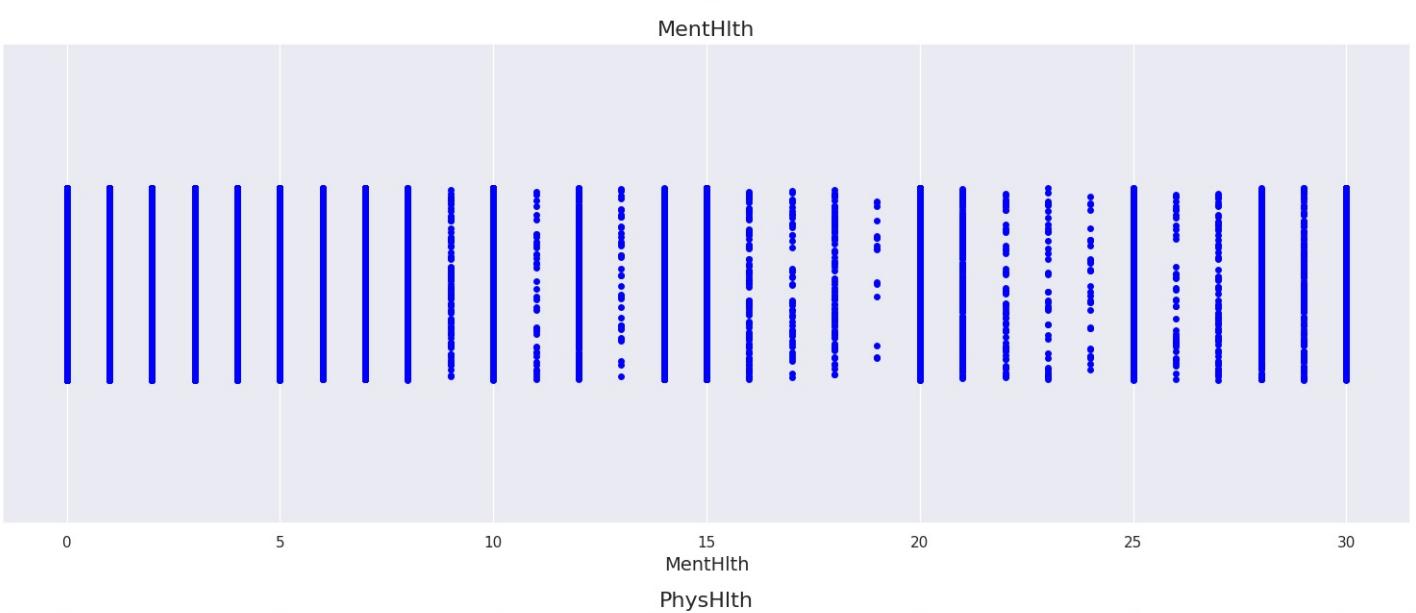
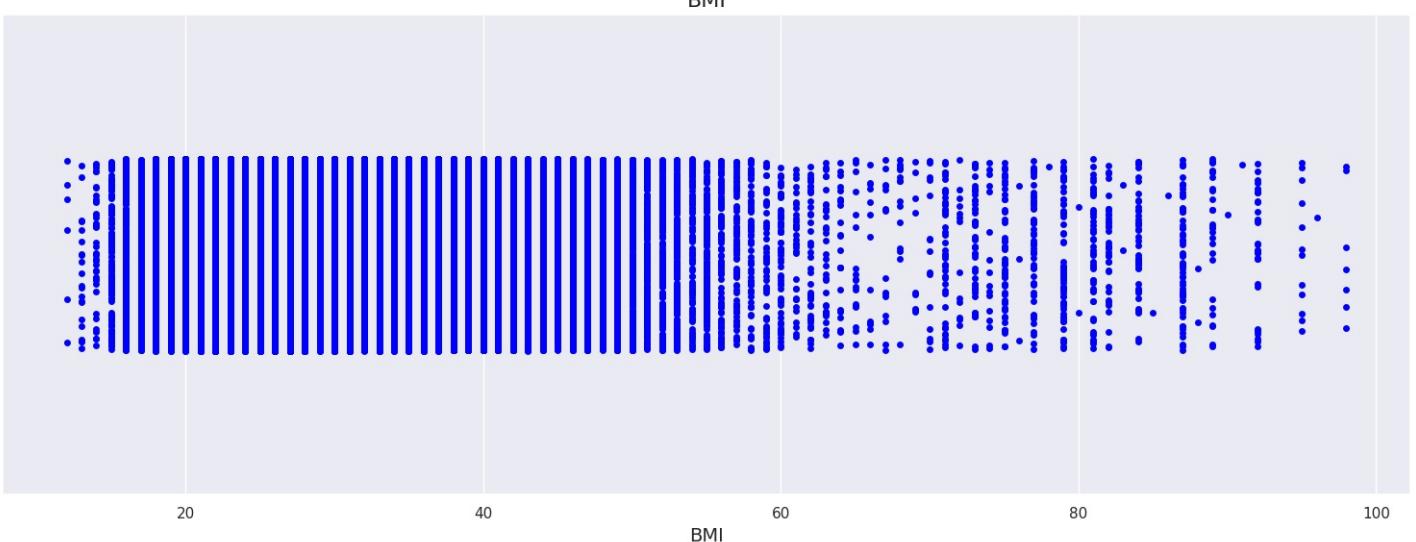
- This plot examines the relationship between age groups and physical health. The median number of poor physical health days is consistent across age groups, with a slight decrease in older age groups (10-13). Outliers are present in each group.

Strip Plot

```
# Set up the matplotlib figure
plt.figure(figsize=(15, 6 * len(numerical)))

# Iterate over each numerical feature to create a strip plot
for index, feature in enumerate(numerical):
    plt.subplot(len(numerical), 1, index + 1) # Creating a subplot for each feature
    sns.stripplot(x=df[feature], color='blue', jitter=0.2) # Adding jitter for better visibility of points
    plt.title(feature, fontsize=16) # title
    plt.xlabel(feature, fontsize=14) # x-label

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



General Observations:

Skewness: Both MentHlth and PhysHlth variables show a left-skewed distribution, with a high concentration of data points at lower values (fewer days of poor health) and a long tail towards higher values.

Outliers and Extremes: All three variables display outliers or extreme values. For BMI, outliers are seen at the very low and very high ends. For MentHlth and PhysHlth, outliers are more evenly distributed but still visible at higher values.

Data Distribution: The strip plots indicate a varied distribution across different ranges for each variable, providing a clear visual of individual data points' spread and density.

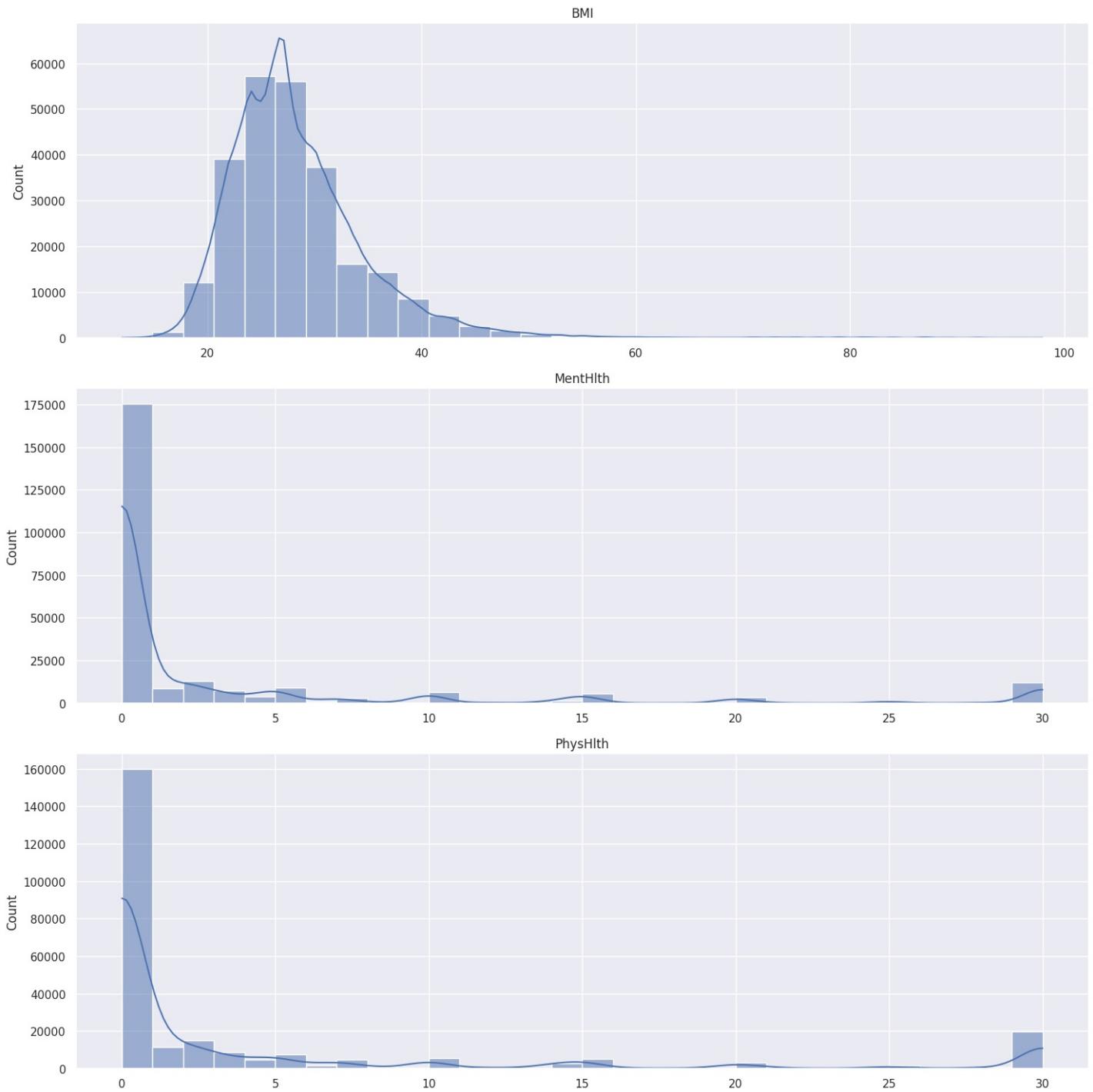
Histogram

Numerical

```
# Set up the matplotlib figure and size
plt.figure(figsize=(15, 5 * len(numerical)))

# Iterate over each numerical feature to create a histogram
for index, feature in enumerate(numerical):
    plt.subplot(len(numerical), 1, index + 1)
    sns.histplot(data=df, x=feature, kde=True, bins=30) # Added KDE plot and adjusted bin count
    plt.title(feature)
    plt.xlabel("") # Remove x-labels as they are redundant with titles
    plt.ylabel("Count") # Ensure y-label is consistently set to 'Count'

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



BMI:

Shows a typical right-skewed distribution with most values in the normal to overweight range. There are fewer extremely high BMI values, which could be clinically significant.

MentHlth and PhysHlth:

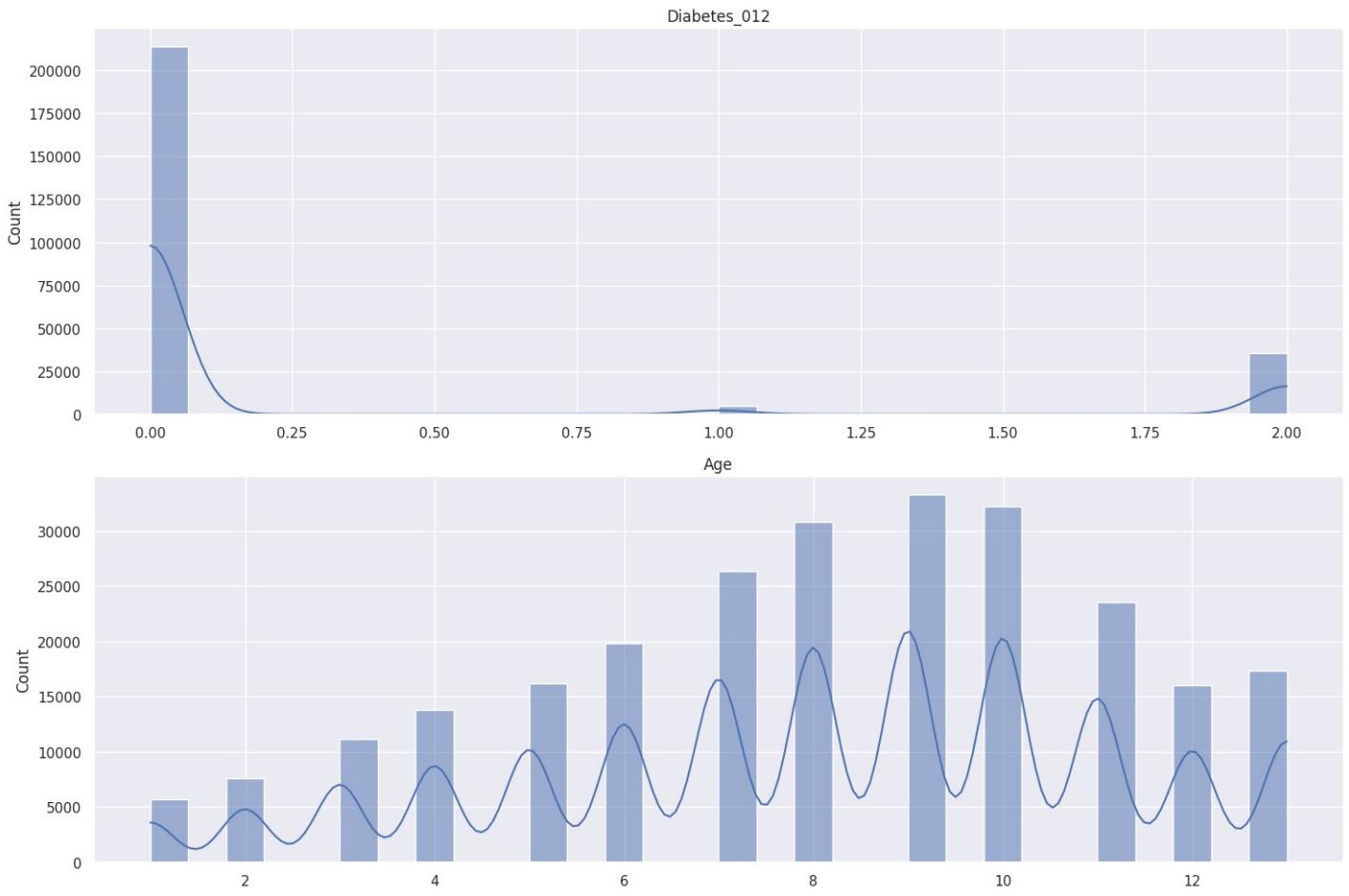
Both variables exhibit a highly right-skewed distribution with a majority of individuals reporting few or no days of poor health. The presence of outliers indicates that some individuals experience extended periods of poor mental and physical health.

Categorical

```
# Set up the matplotlib figure and size
plt.figure(figsize=(15, 5 * len(categorical)))

# Iterate over each numerical feature to create a histogram
for index, feature in enumerate(categorical):
    plt.subplot(len(categorical), 1, index + 1)
    sns.histplot(data=df, x=feature, kde=True, bins=30) # Added KDE plot and adjusted bin count
    plt.title(feature)
    plt.xlabel("") # Remove x-labels as they are redundant with titles
    plt.ylabel('Count') # Ensure y-label is consistently set to 'Count'

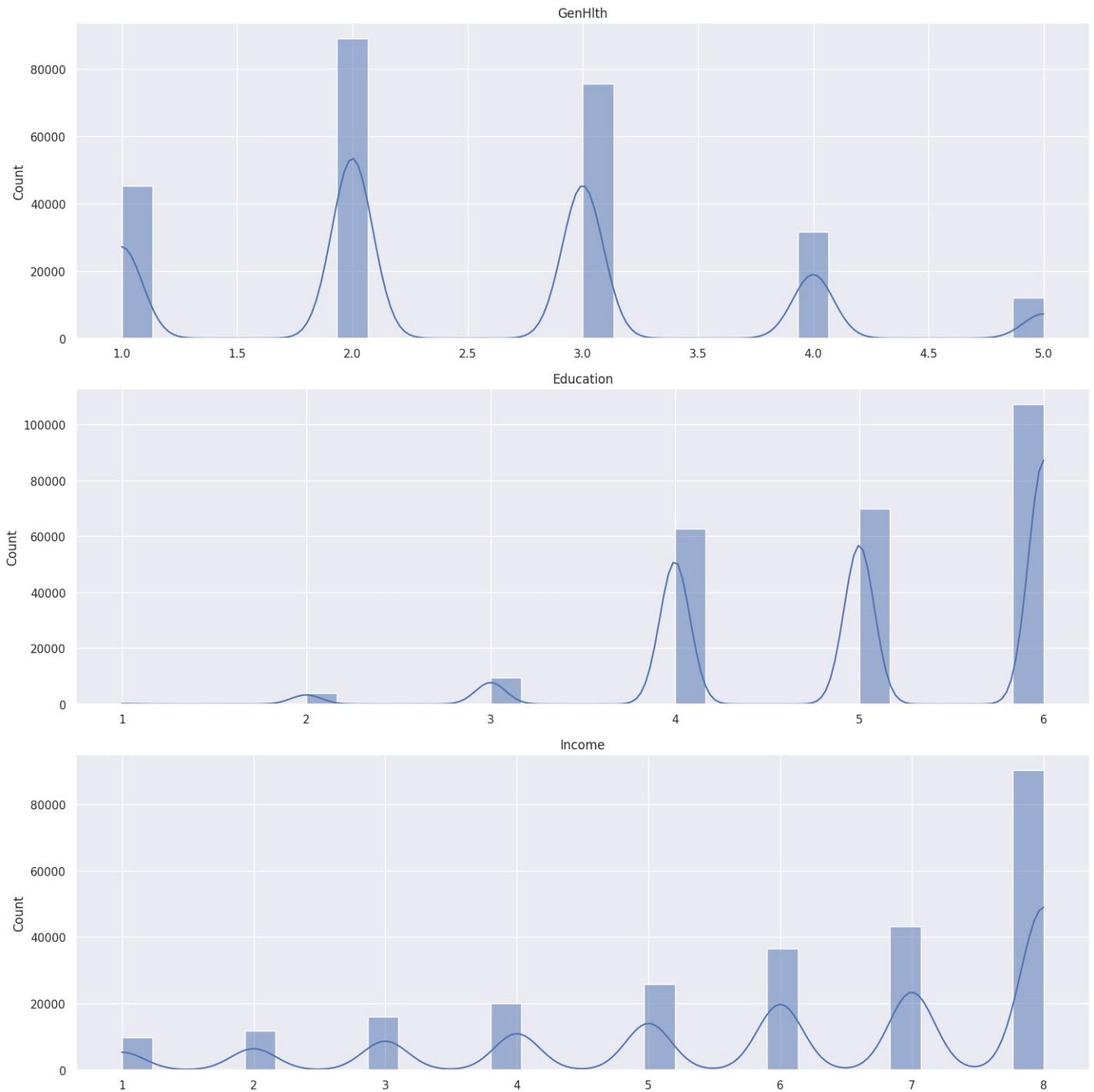
plt.tight_layout()
plt.show()
```



```
# Set up the matplotlib figure and size
plt.figure(figsize=(15, 5 * len(ordinal)))

# Iterate over each numerical feature to create a histogram
for index, feature in enumerate(ordinal):
    plt.subplot(len(ordinal), 1, index + 1)
    sns.histplot(data=df, x=feature, kde=True, bins=30) # Added KDE plot and adjusted bin count
    plt.title(feature)
    plt.xlabel("") # Remove x-labels as they are redundant with titles
    plt.ylabel("Count") # Ensure y-label is consistently set to 'Count'

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



GenHlth Distribution:

The plot shows the distribution of general health ratings (GenHlth), which are categorized from 1 (excellent) to 5 (poor). The most common ratings are 2 and 3, indicating that many individuals rate their health as "very good" or "good". There are fair (4), or poor (5).

Education Distribution:

The plot shows the distribution of education levels, which range from 1 to 6. The highest count is for education level 6, indicating that many individuals in the dataset have a higher education level. Other peaks are observed at levels 4 and 5, while levels 1, 2, and 3 have very low counts.

Income Distribution:

The plot shows the distribution of income levels, which range from 1 to 8. There are noticeable peaks at levels 4, 6, and 8, suggesting that these income levels are more common in the dataset. The distribution has smaller peaks at levels 1, 2, 3, 5, and 7.

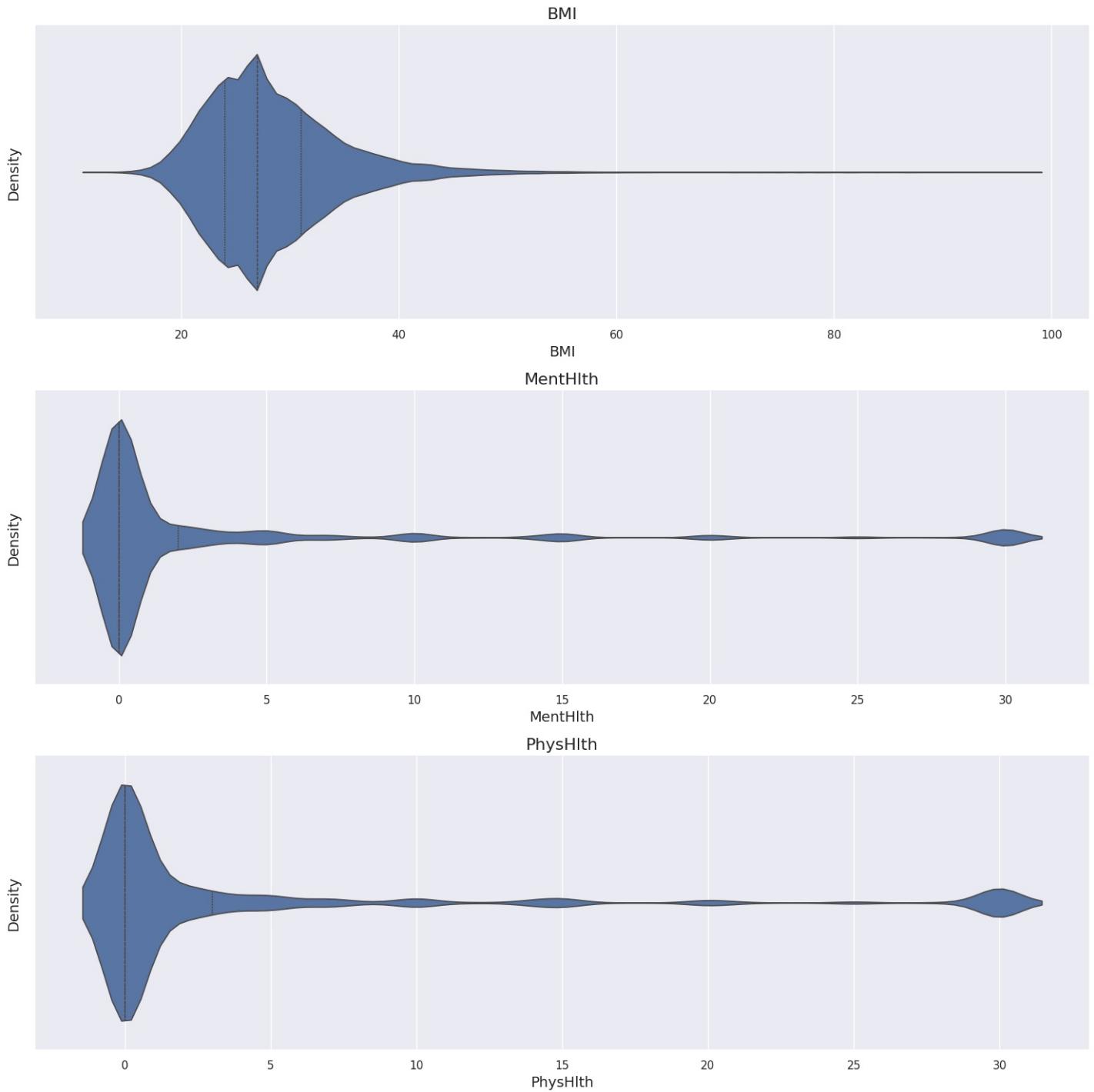
Violin Plots

Numerical

```
# Set up the matplotlib figure and size
plt.figure(figsize=(15, 5 * len(numerical)))

# Iterate over each numerical feature to create a violin plot
for index, feature in enumerate(numerical):
    plt.subplot(len(numerical), 1, index + 1) # Create a subplot for each feature
    sns.violinplot(data=df, x=feature, inner='quartile') # inner='quartile' adds quartiles
    plt.title(feature, fontsize=16) # Setting the title with a larger font size
    plt.xlabel(feature, fontsize=14) # Setting the x-label with a readable font size
    plt.ylabel('Density', fontsize=14) # Setting the y-label to 'Density'

plt.tight_layout() # Adjusting subplots to fit into the figure area nicely
plt.show() # Displaying the plots
```



BMI: Most individuals have a BMI between 25-30, with density tapering off at higher and lower values.

MentHlth: Most individuals report 0 poor mental health days, with decreasing density as the number of poor mental health days increases.

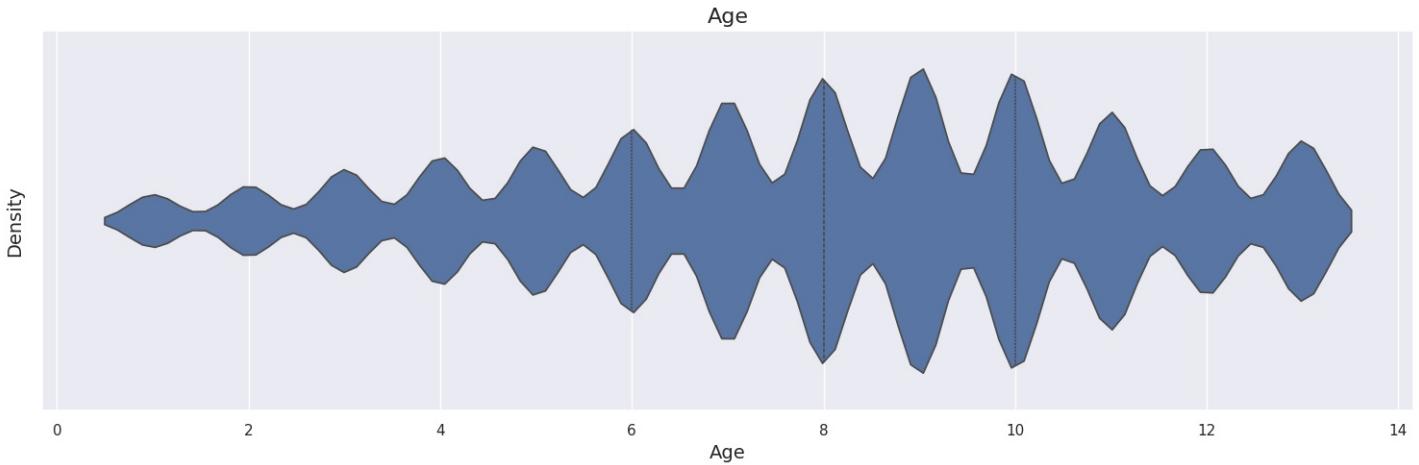
PhysHlth: Similar to MentHlth, most individuals report 0 poor physical health days, with density decreasing as the number of poor physical health days increases.

Categorical

```
# Set up the matplotlib figure and size
plt.figure(figsize=(15, 5 * len(categorical)))

# Iterate over each numerical feature to create a violin plot
for index, feature in enumerate(categorical):
    plt.subplot(len(categorical), 1, index + 1) # Create a subplot for each feature
    sns.violinplot(data=df, x=feature, inner='quartile') # inner='quartile' adds quartiles
    plt.title(feature, fontsize=16) # Setting the title with a larger font size
    plt.xlabel(feature, fontsize=14) # Setting the x-label with a readable font size
    plt.ylabel('Density', fontsize=14) # Setting the y-label to 'Density'

plt.tight_layout() # Adjusting subplots to fit into the figure area nicely
plt.show() # Displaying the plots
```



Diabetes_012: High density at 0 (no diabetes), with lower densities at 1 (prediabetes) and 2 (diabetes).

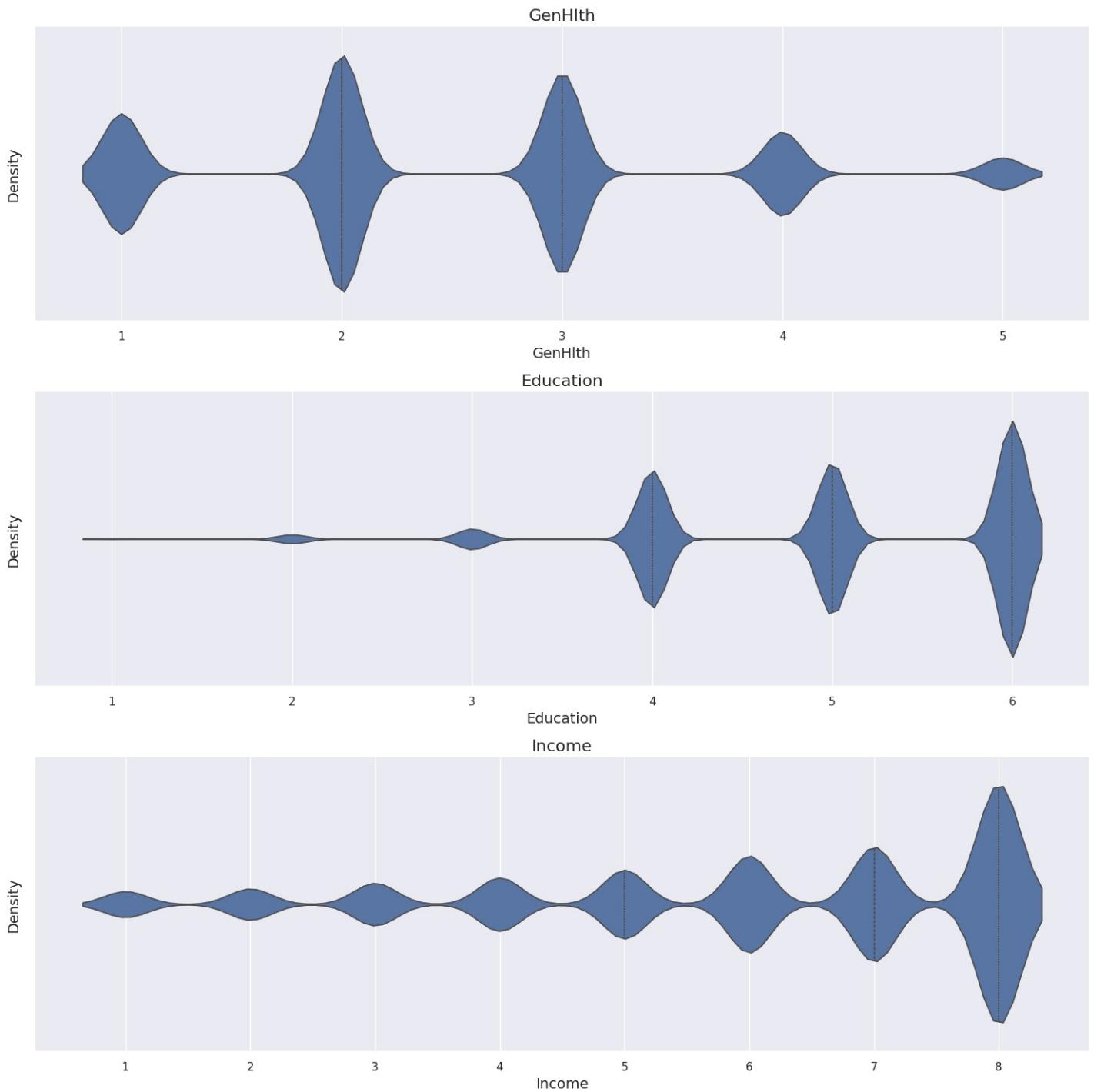
Age: Peaks at age groups 4, 8, and 10, indicating higher representation in these groups.

Ordinal

```
# Set up the matplotlib figure and size
plt.figure(figsize=(15, 5 * len(ordinal)))

# Iterate over each numerical feature to create a violin plot
for index, feature in enumerate(ordinal):
    plt.subplot(len(ordinal), 1, index + 1) # Create a subplot for each feature
    sns.violinplot(data=df, x=feature, inner='quartile') # inner='quartile' adds quartiles
    plt.title(feature, fontsize=16) # Setting the title with a larger font size
    plt.xlabel(feature, fontsize=14) # Setting the x-label with a readable font size
    plt.ylabel("Density", fontsize=14) # Setting the y-label to 'Density'

plt.tight_layout() # Adjusting subplots to fit into the figure area nicely
plt.show() # Displaying the plots
```



GenHlth: Peaks at 2 (very good) and 3 (good), with fewer individuals at 1 (excellent), 4 (fair), and 5 (poor).

Education: Highest density at 6 (highest education level), with peaks at 4 and 5, and lower densities at 1, 2, and 3.

Income: Peaks at 4, 6, and 8, indicating these income levels are more common, with lower densities at other levels.

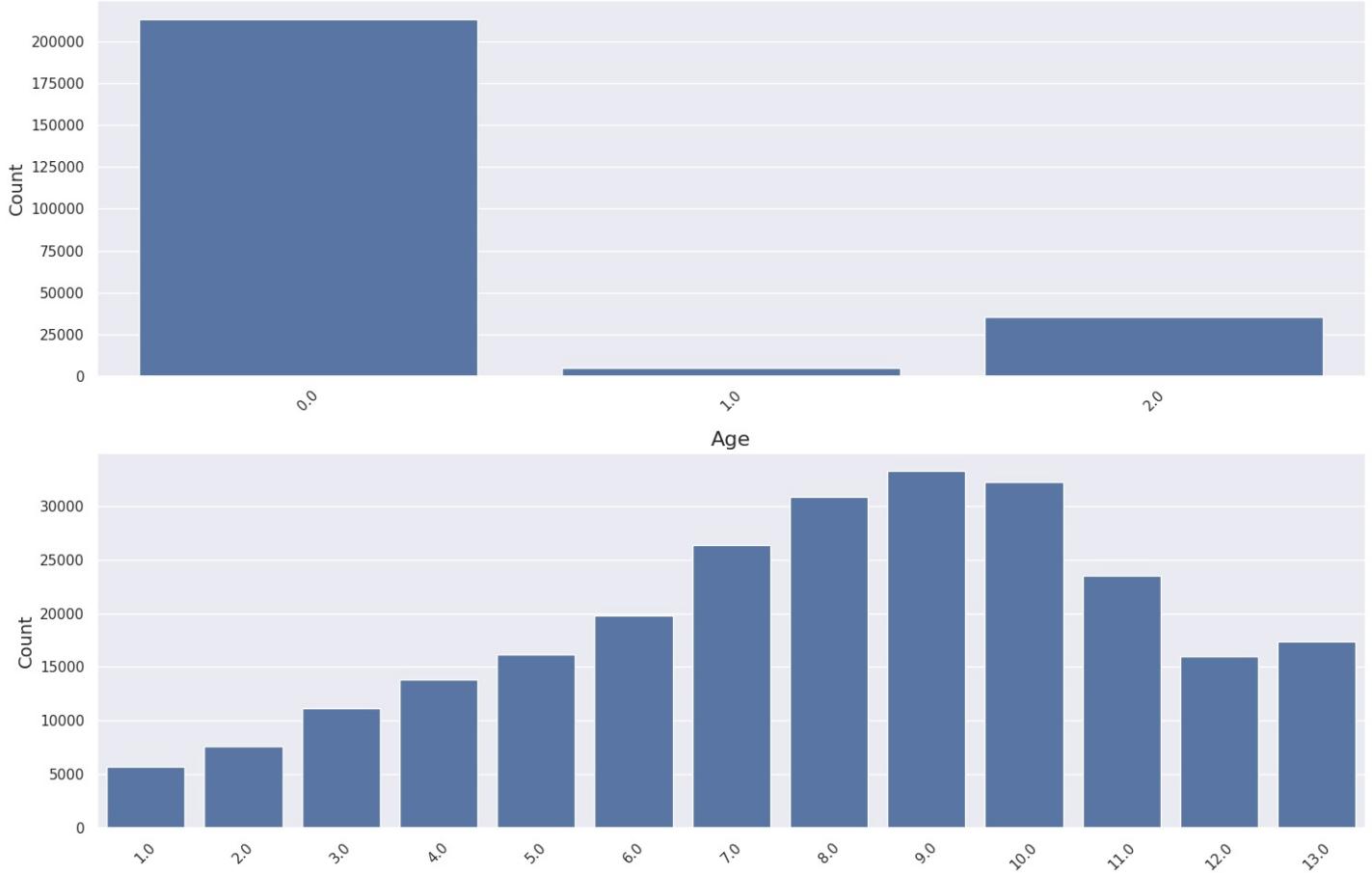
Categorical Variables - Countplot

```
# Set up the matplotlib figure and size
plt.figure(figsize=(15, 5 * len(categorical))) # Adjust the figure size to fit all plots

# Iterate over each categorical feature to create a count plot
for index, feature in enumerate(categorical):
    plt.subplot(len(categorical), 1, index + 1)
    sns.countplot(data=df, x=feature)
    plt.title(feature, fontsize=16)
    plt.xlabel("") # Remove x-labels as they are redundant with titles
    plt.ylabel('Count', fontsize=14) # Ensure y-label is consistently set to 'Count'
    plt.xticks(rotation=45) # Rotate x-axis labels by 45 degrees for readability

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

Diabetes_012

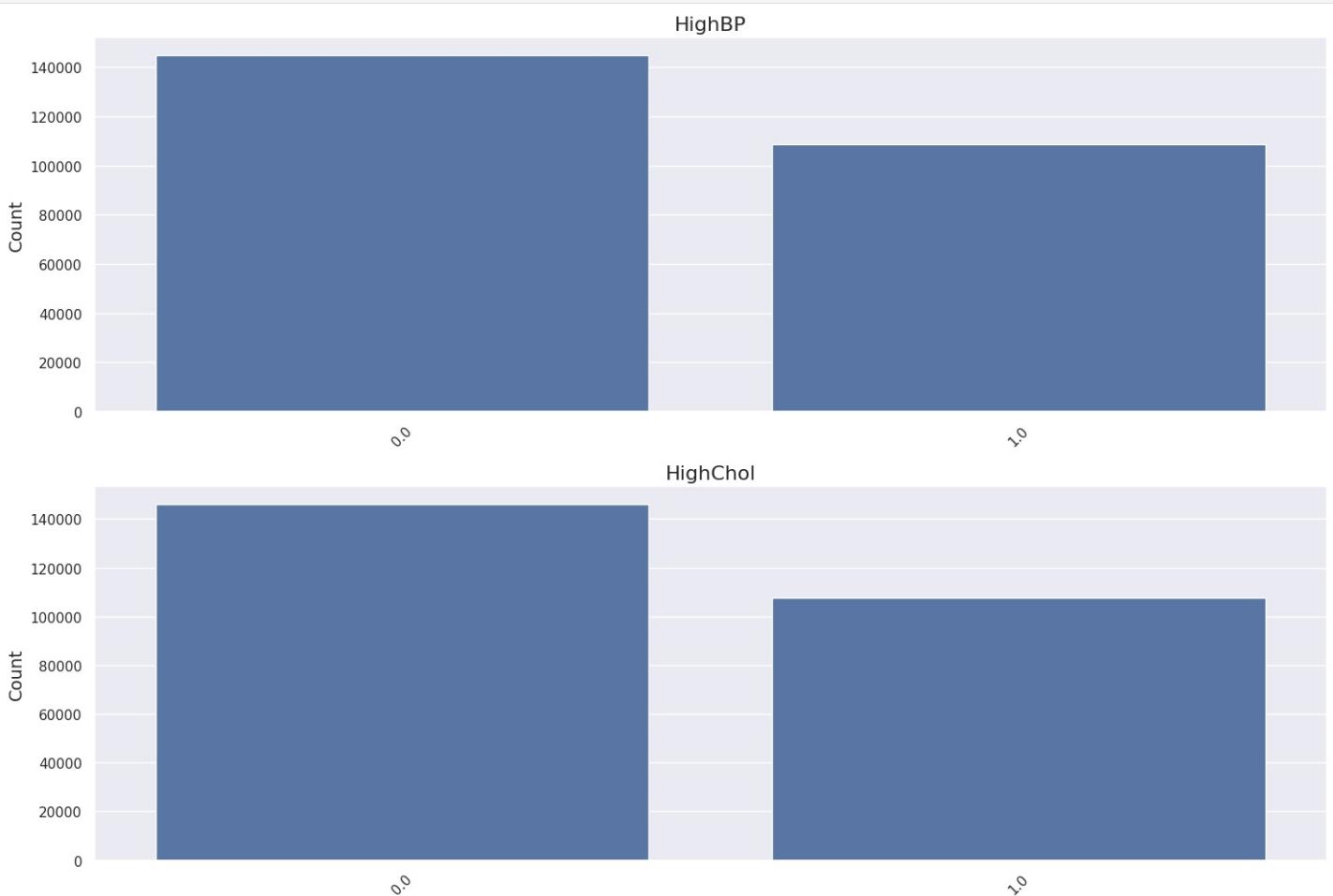


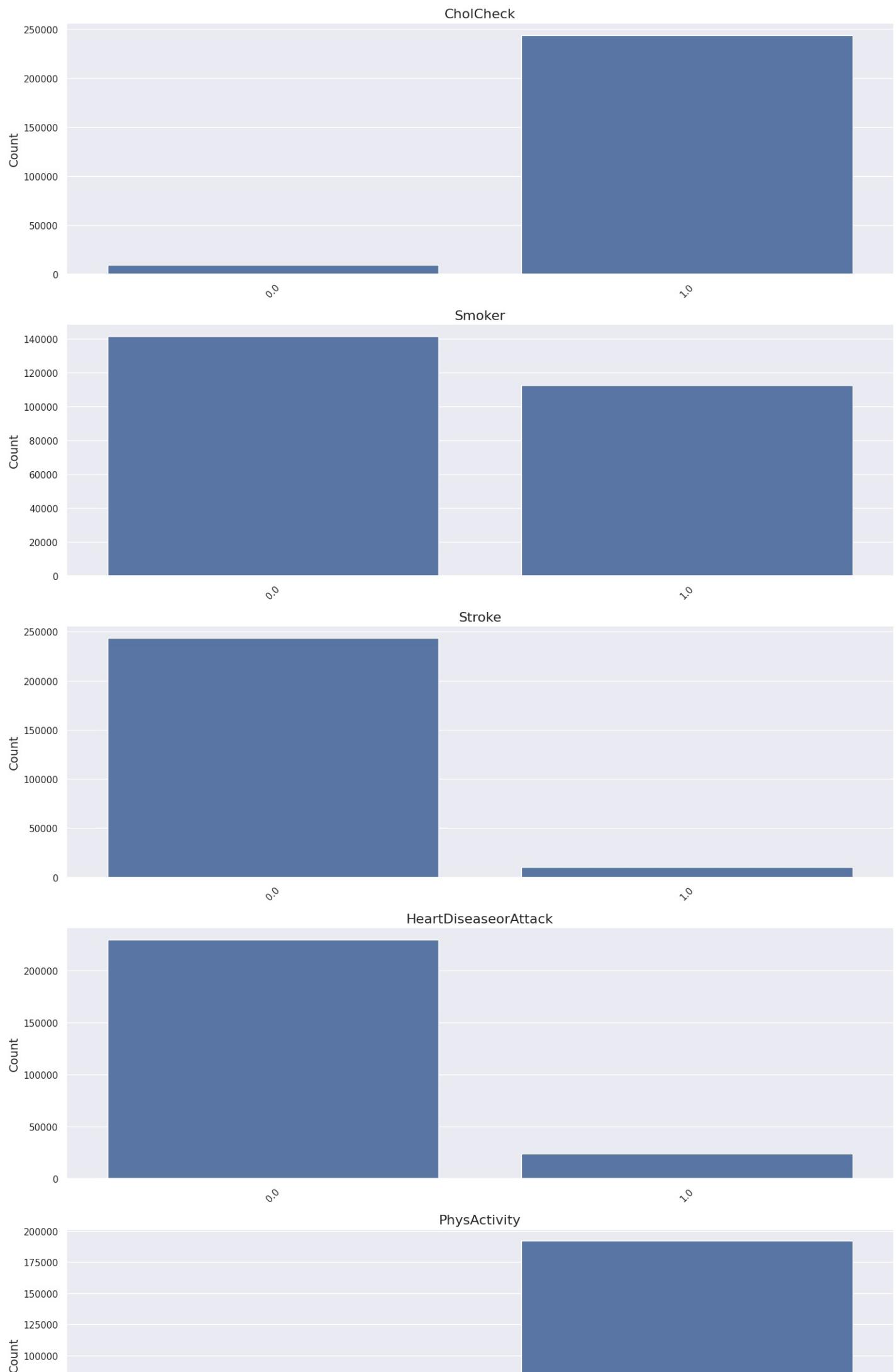
Binary Values - Countplot

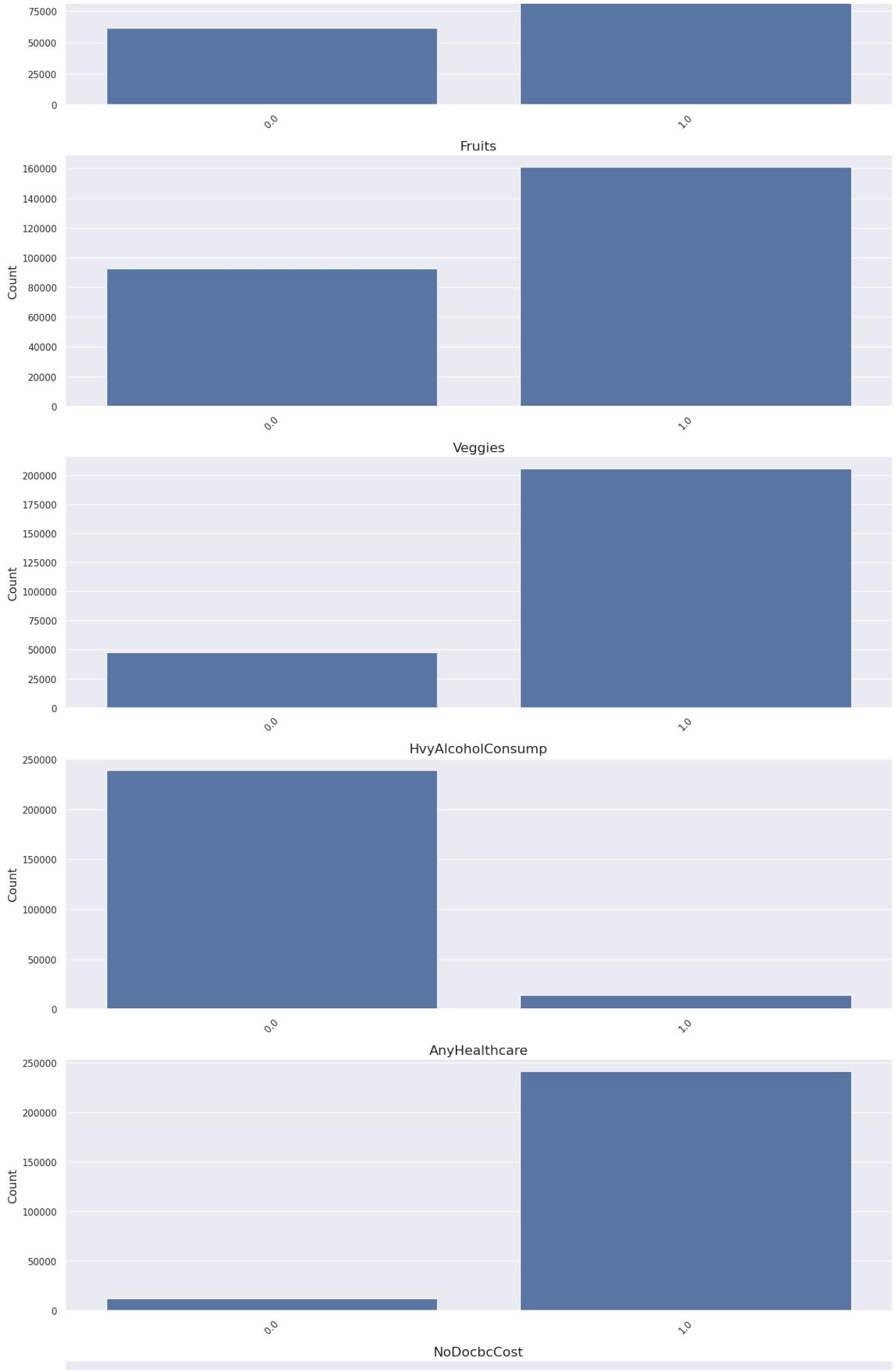
```
# Set up the matplotlib figure and size
plt.figure(figsize=(15, 5 * len(binary))) # Adjust the figure size to fit all plots

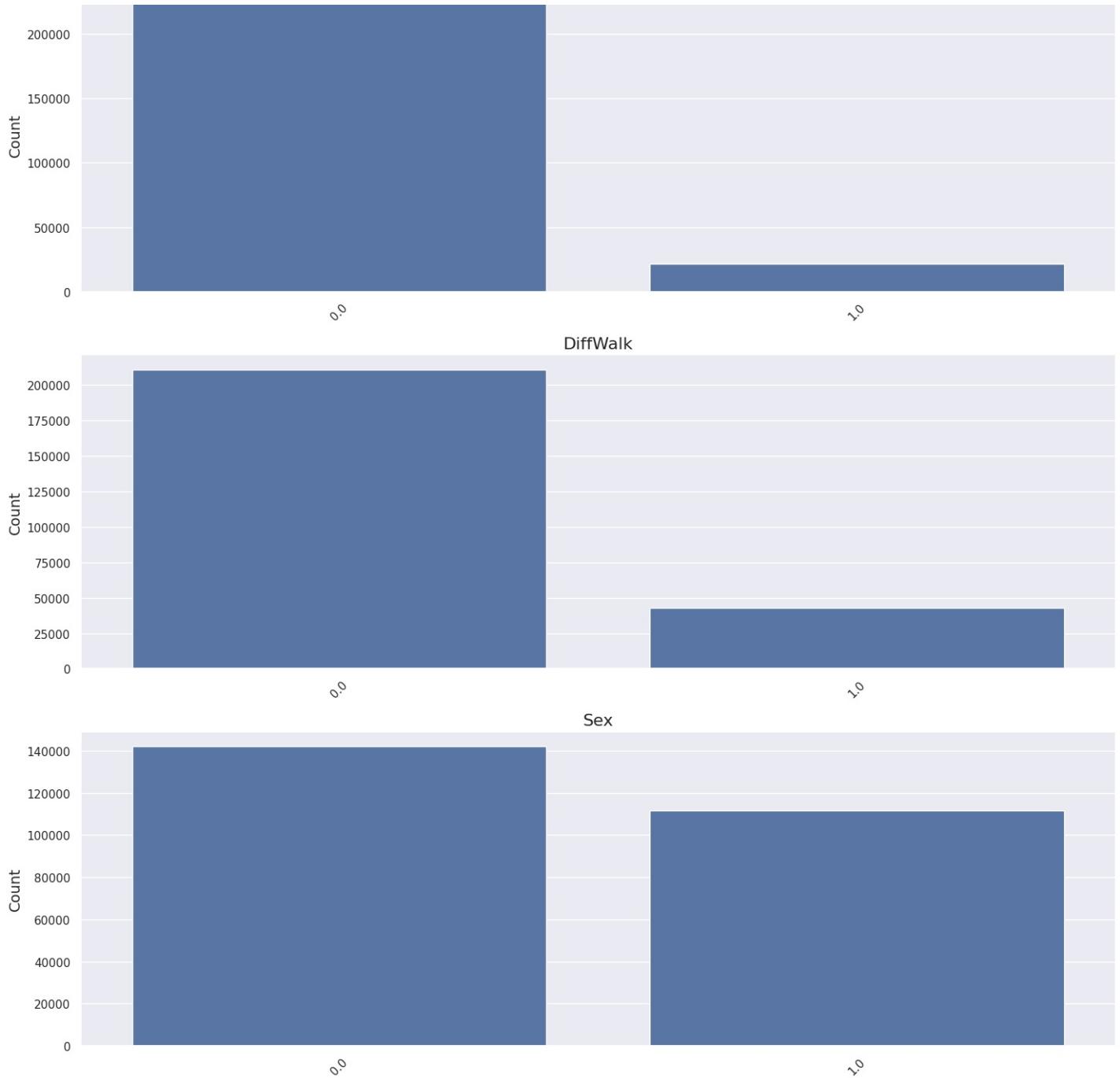
# Iterate over each categorical feature to create a count plot
for index, feature in enumerate(binary):
    plt.subplot(len(binary), 1, index + 1)
    sns.countplot(data=df, x=feature)
    plt.title(feature, fontsize=16)
    plt.xlabel("") # Remove x-labels as they are redundant with titles
    plt.ylabel("Count", fontsize=14) # Ensure y-label is consistently set to 'Count'
    plt.xticks(rotation=45) # Rotate x-axis labels by 45 degrees for readability

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







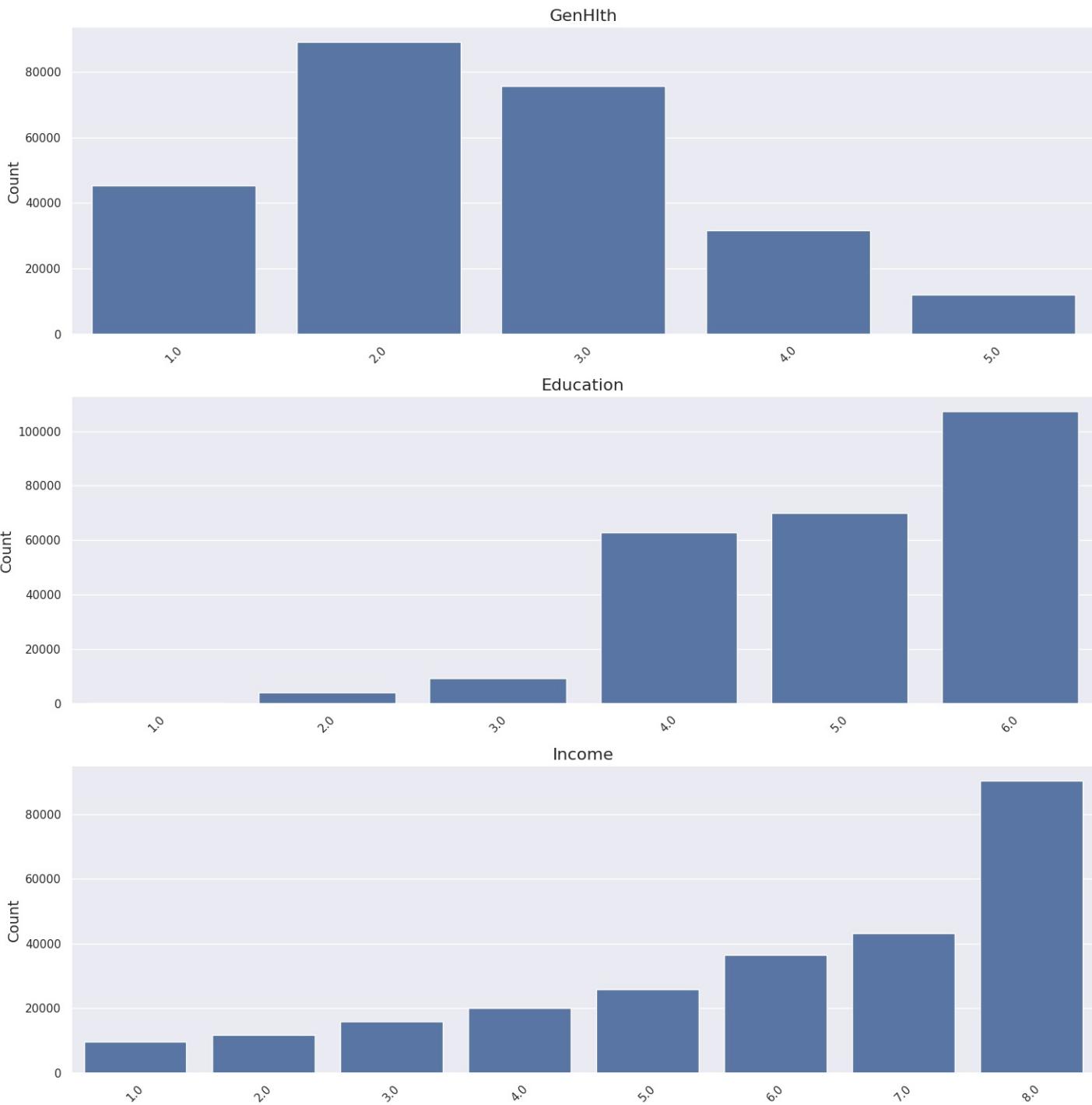


Ordinal Values - Countplot

```
# Set up the matplotlib figure and size
plt.figure(figsize=(15, 5 * len(ordinal))) # Adjust the figure size to fit all plots

# Iterate over each categorical feature to create a count plot
for index, feature in enumerate(ordinal):
    plt.subplot(len(ordinal), 1, index + 1)
    sns.countplot(data=df, x=feature)
    plt.title(feature, fontsize=16)
    plt.xlabel("") # Remove x-labels as they are redundant with titles
    plt.ylabel('Count', fontsize=14) # Ensure y-label is consistently set to 'Count'
    plt.xticks(rotation=45) # Rotate x-axis labels by 45 degrees for readability

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



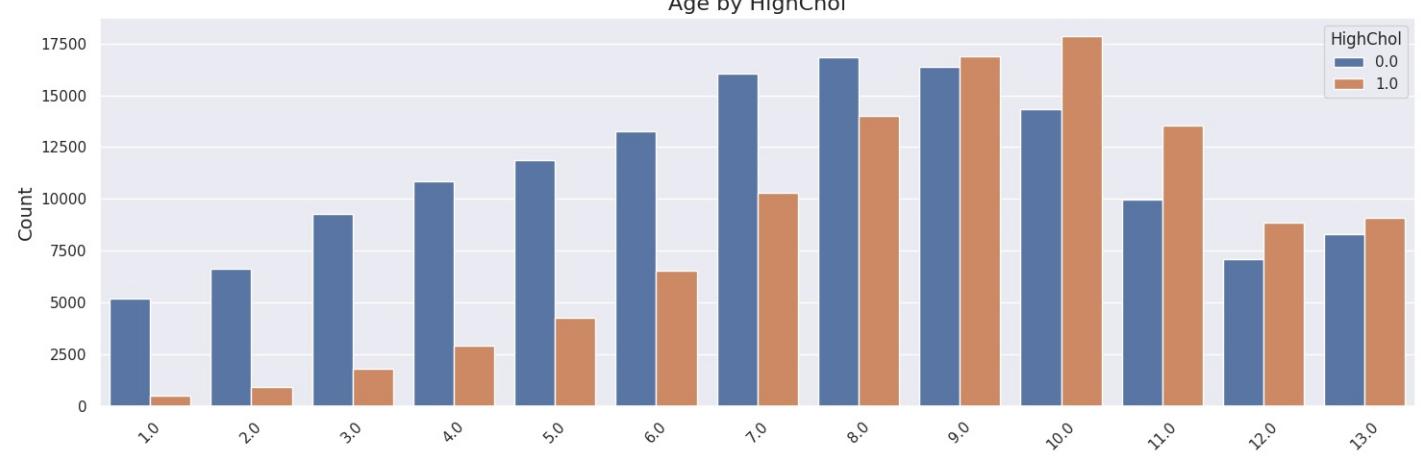
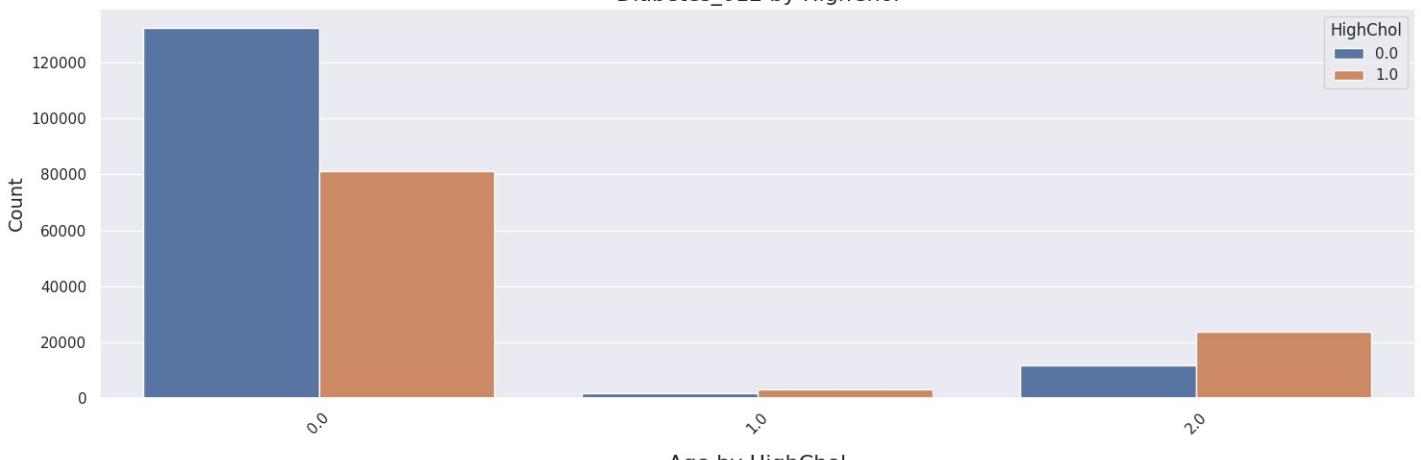
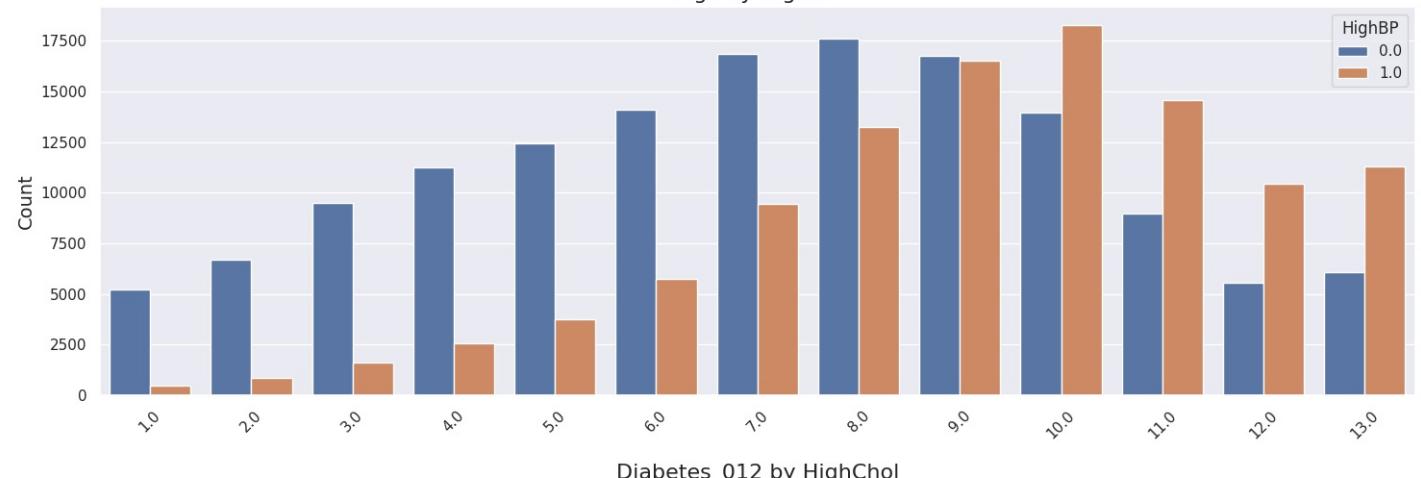
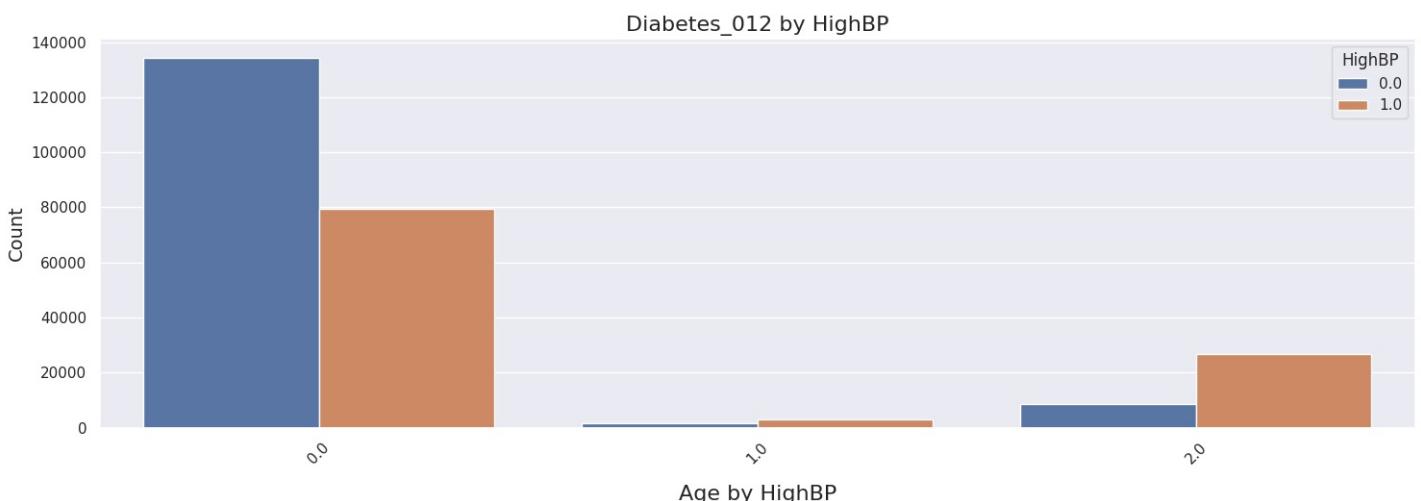
Histplot with hue

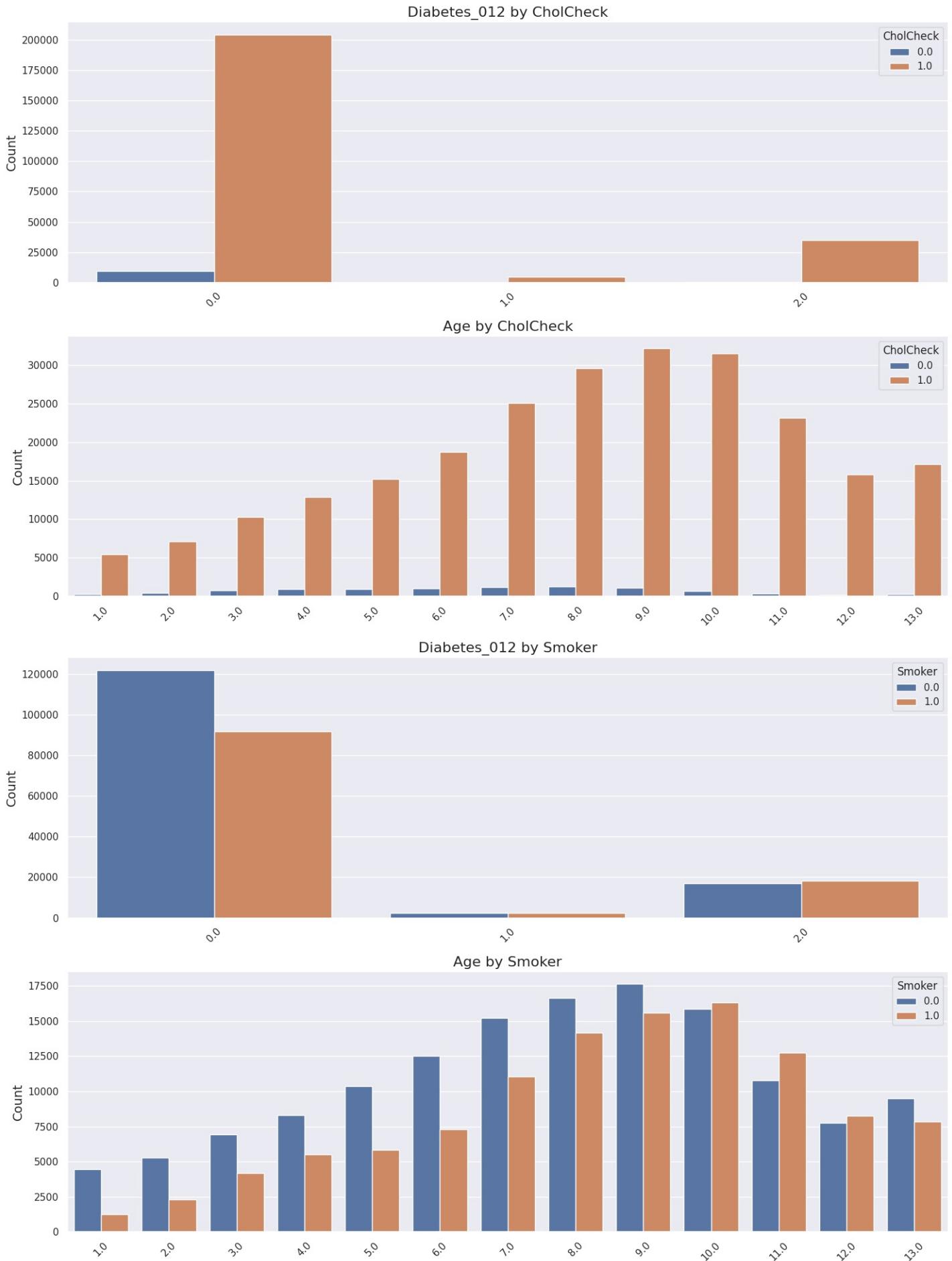
Categorical vs Binary

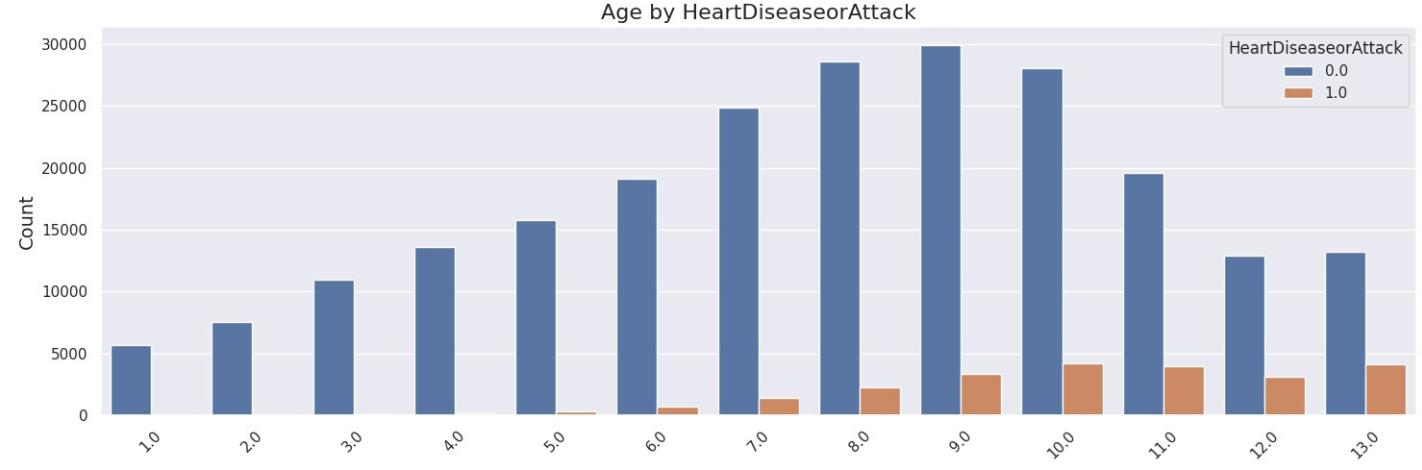
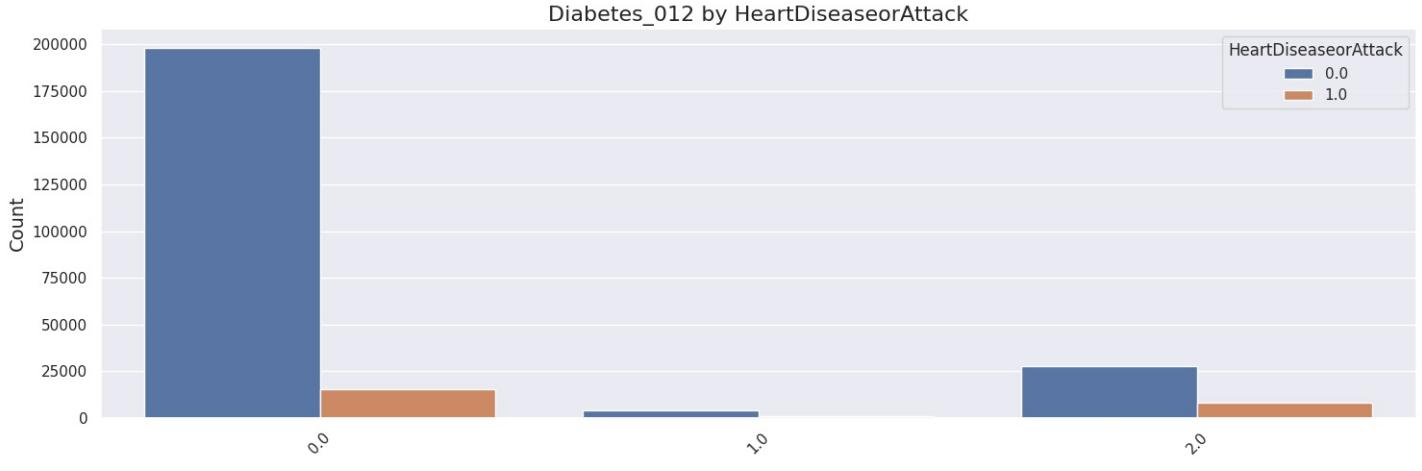
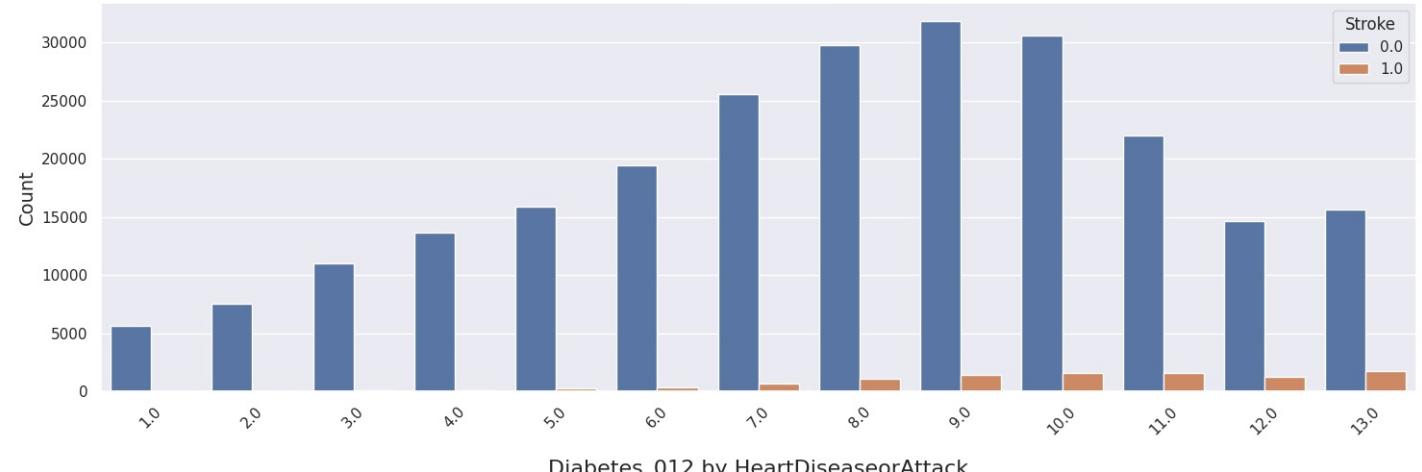
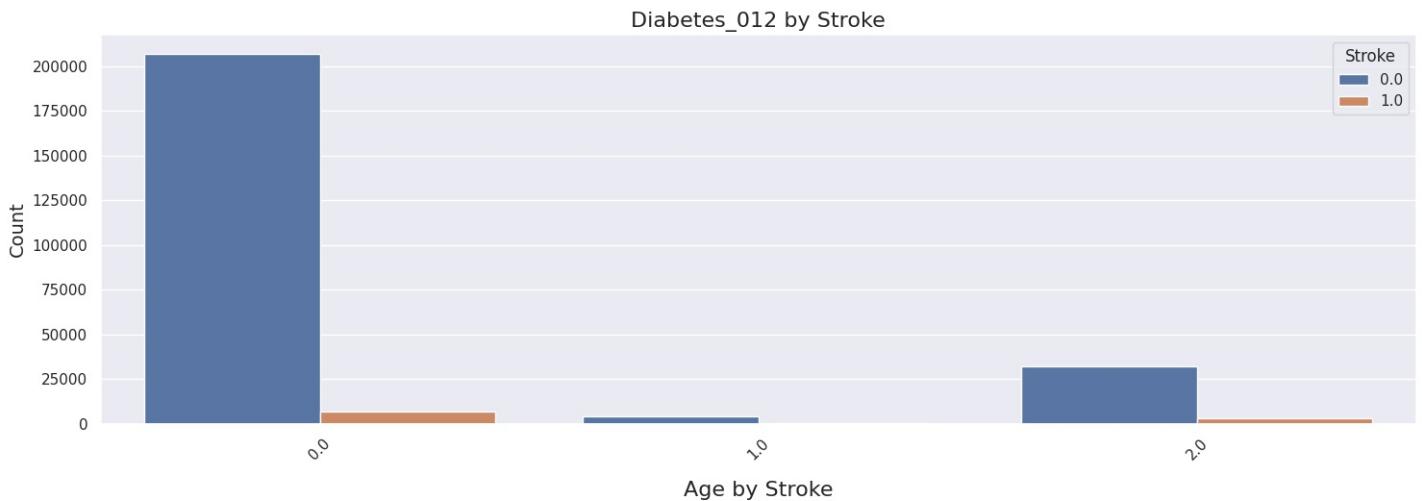
```
# Iterate over each binary feature as hue
for hue_feature in binary:
    # Set up the matplotlib figure and size
    plt.figure(figsize=(15, len(categorical) * 5))

    # Iterate over each categorical feature to create a count plot with hue
    for index, feature in enumerate(categorical):
        plt.subplot(len(categorical), 1, index + 1)
        sns.countplot(data=df, x=feature, hue=hue_feature)
        plt.title(f'{feature} by {hue_feature}', fontsize=16)
        plt.xlabel('') # Remove x-labels as they are redundant with titles
        plt.ylabel('Count', fontsize=14) # Ensure y-label is consistently set to 'Count'
        plt.xticks(rotation=45) # Rotate x-axis labels by 45 degrees for readability
        plt.legend(title=hue_feature, loc='upper right') # Add legend

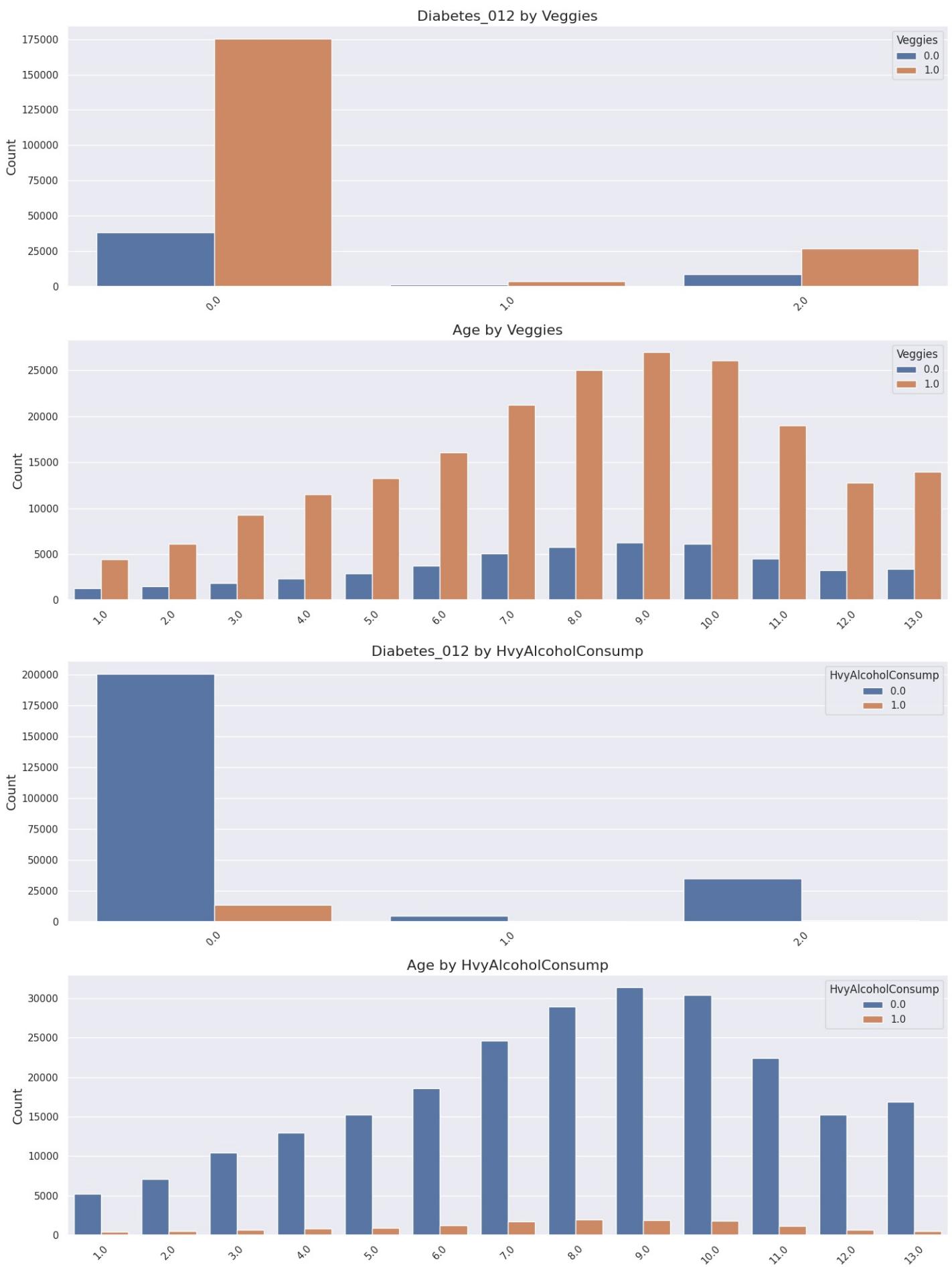
    plt.tight_layout()
plt.show()
```



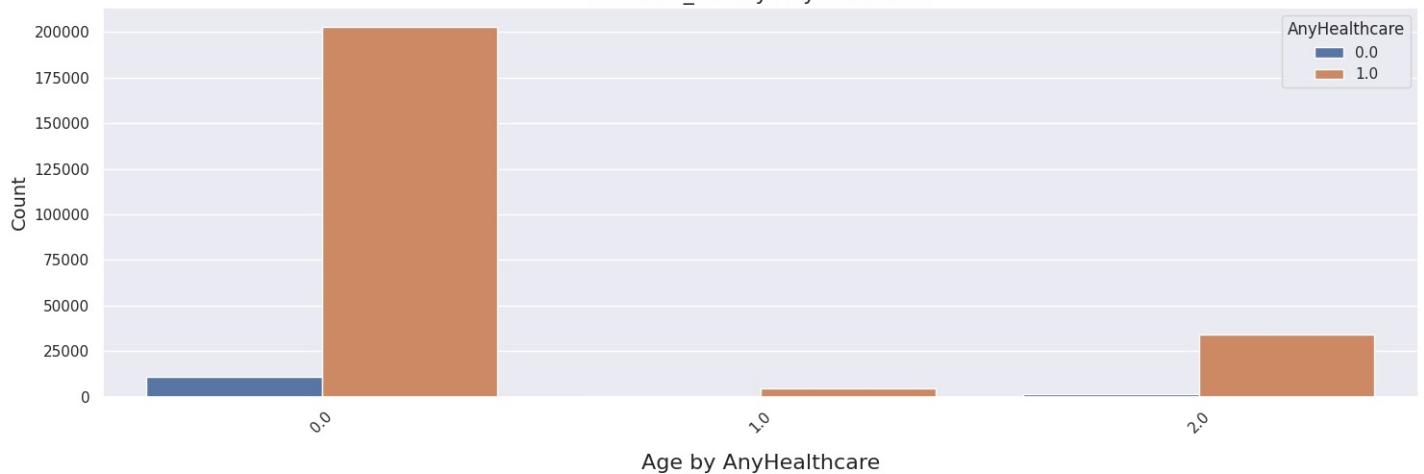




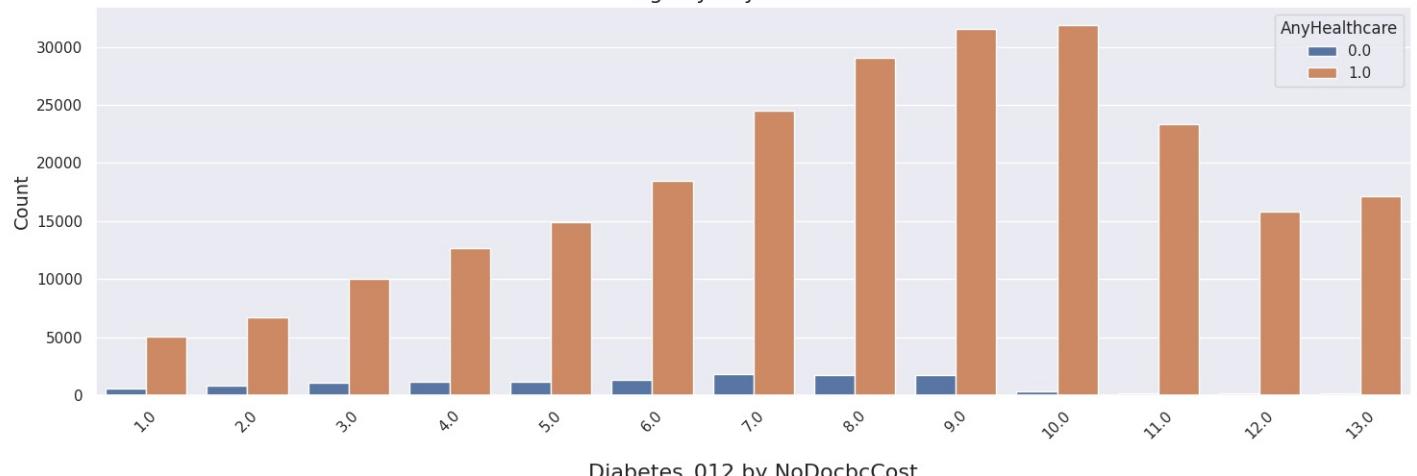




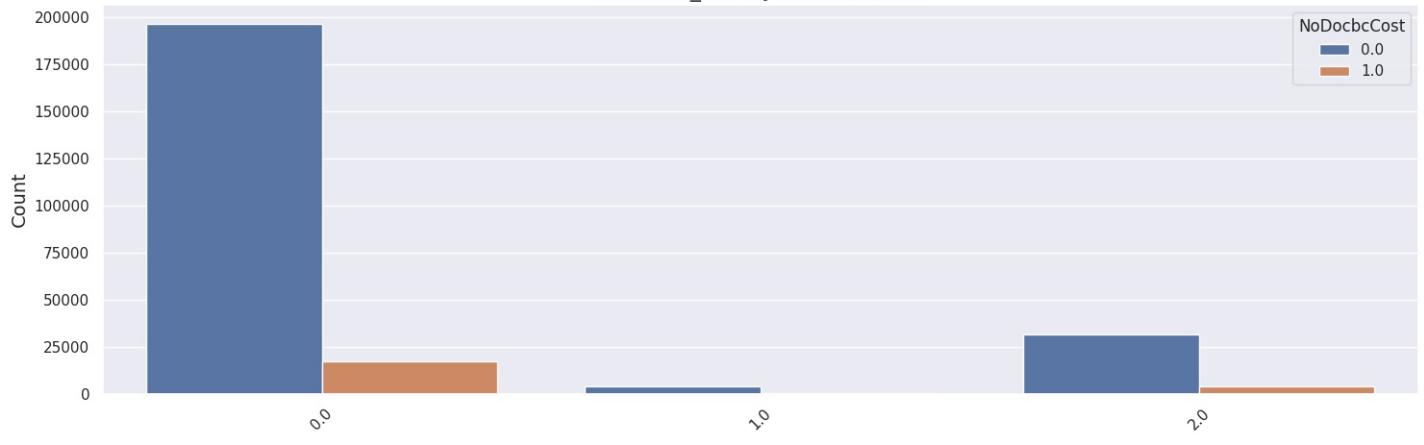
Diabetes_012 by AnyHealthcare



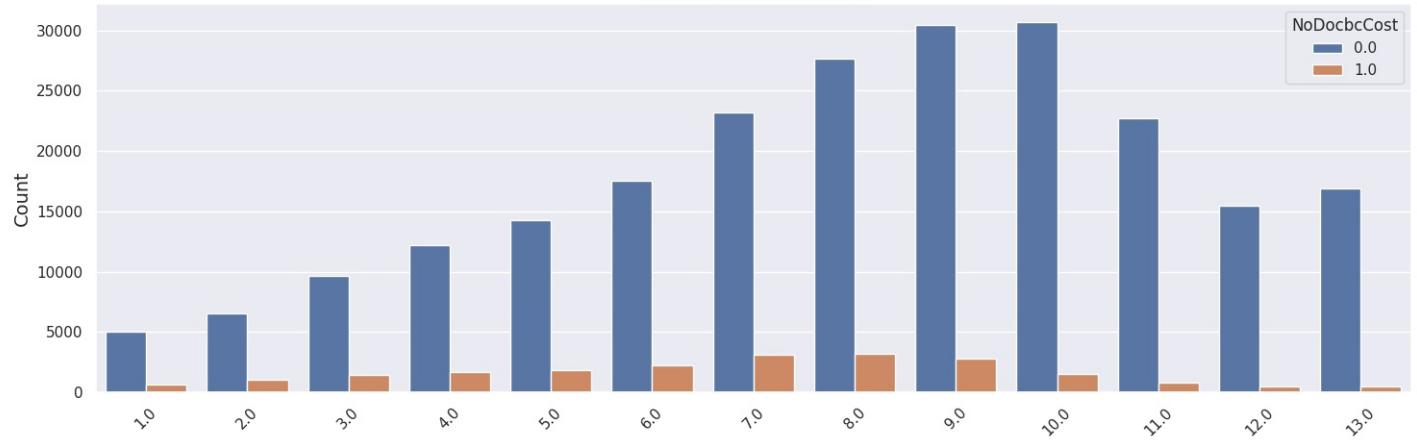
Age by AnyHealthcare



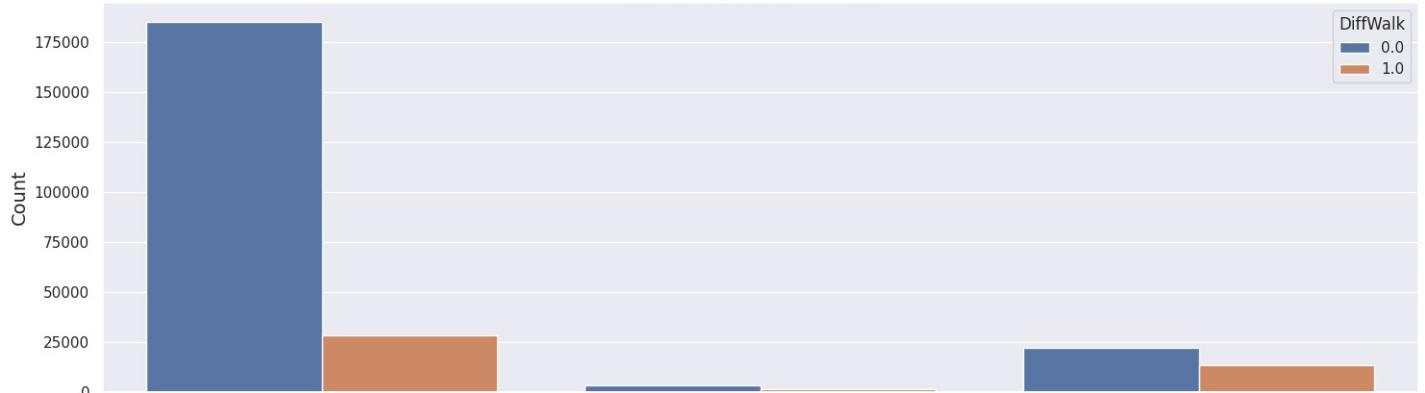
Diabetes_012 by NoDocbcCost



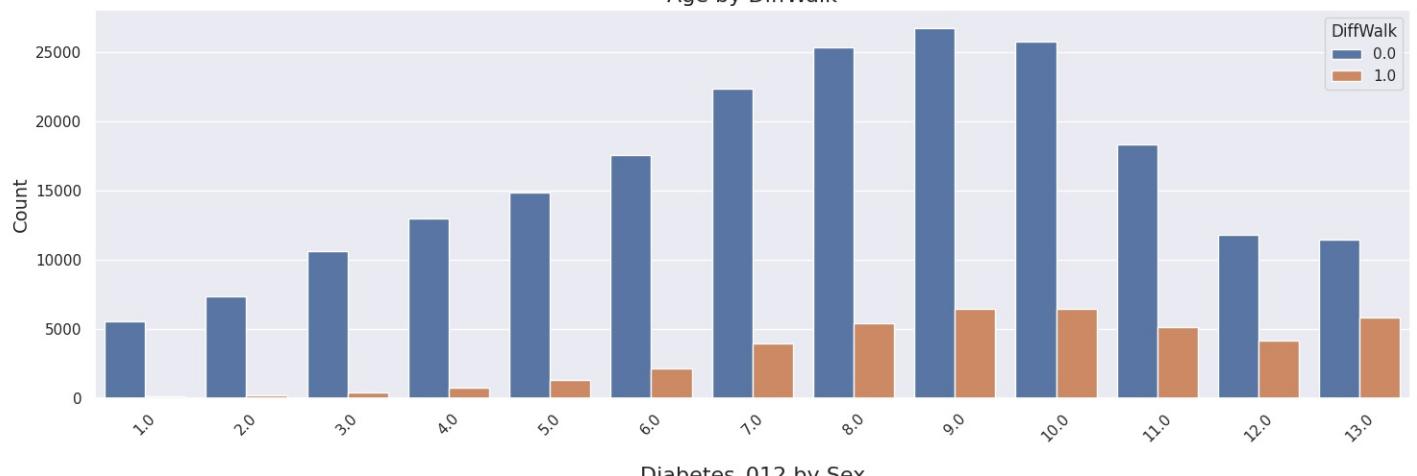
Age by NoDocbcCost



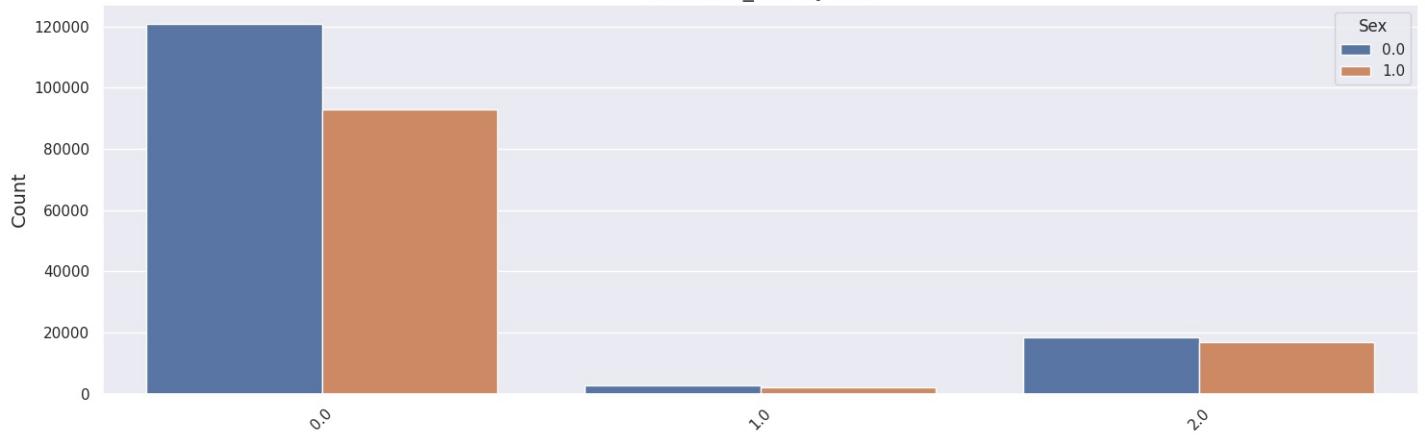
Diabetes_012 by DiffWalk



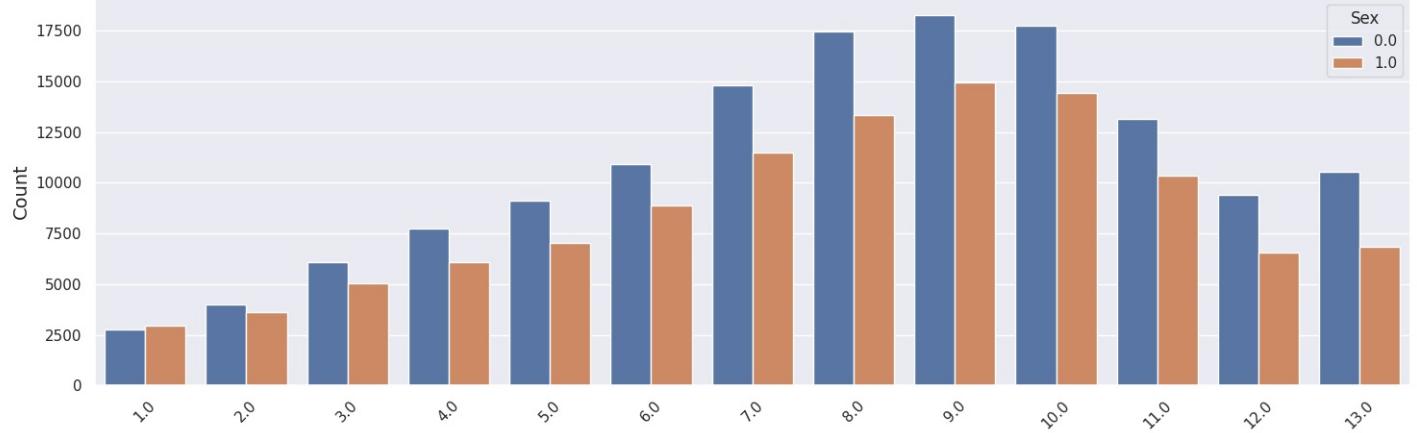
Age by DiffWalk



Diabetes_012 by Sex



Age by Sex



Diabetes_012 Distribution:

The majority of individuals have no diabetes (0), with a very small proportion of the population in the prediabetes (1) and diabetes (2) categories. This indicates that most of the dataset's individuals do not have diabetes.

Age Distribution:

The age groups are coded numerically from 1 to 13, with each group representing different age ranges. The counts fluctuate across these age groups, with peaks observed in groups 4, 8, and 10, indicating that these age ranges have higher representation in the dataset.

Line Plot

BMI

```
# Define the numerical feature for the y axis for BMI
numerical_feature = 'BMI'
```

```

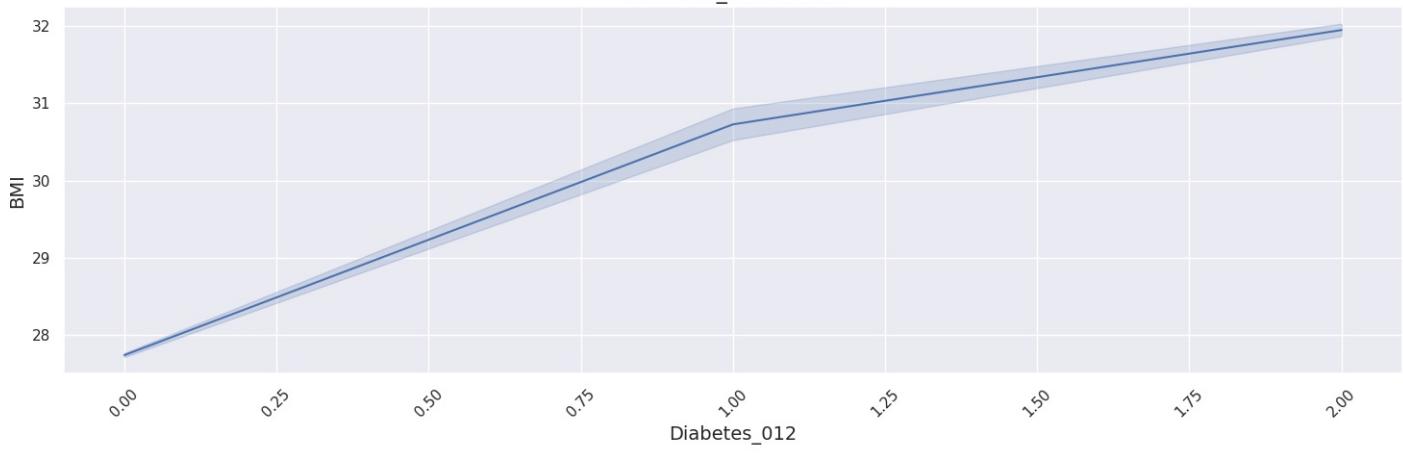
# Set up the matplotlib figure and size
plt.figure(figsize=(15, len(categorical) * 5))

# Iterate over each categorical feature to create a line plot
for index, feature in enumerate(categorical):
    plt.subplot(len(categorical), 1, index + 1)
    sns.lineplot(data=df, x=feature, y=numerical_feature)
    plt.title(f'{feature} vs {numerical_feature}', fontsize=16)
    plt.xlabel(feature, fontsize=14)
    plt.ylabel(numerical_feature, fontsize=14)
    plt.xticks(rotation=45) # Rotate x-axis labels by 45 degrees for readability

plt.tight_layout()
plt.show()

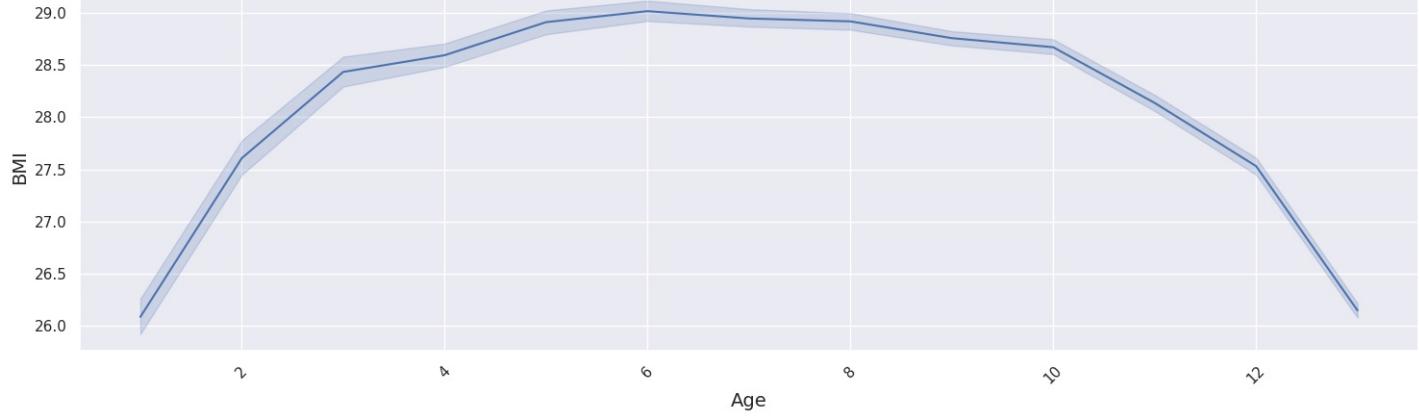
```

Diabetes_012 vs BMI



Diabetes_012

Age vs BMI



Diabetes_012 vs BMI: BMI increases with higher diabetes status, from 28 (no diabetes) to 32 (diabetes).

Age vs BMI: BMI peaks in middle age (around age group 6) and decreases in older age groups.

Mental Health

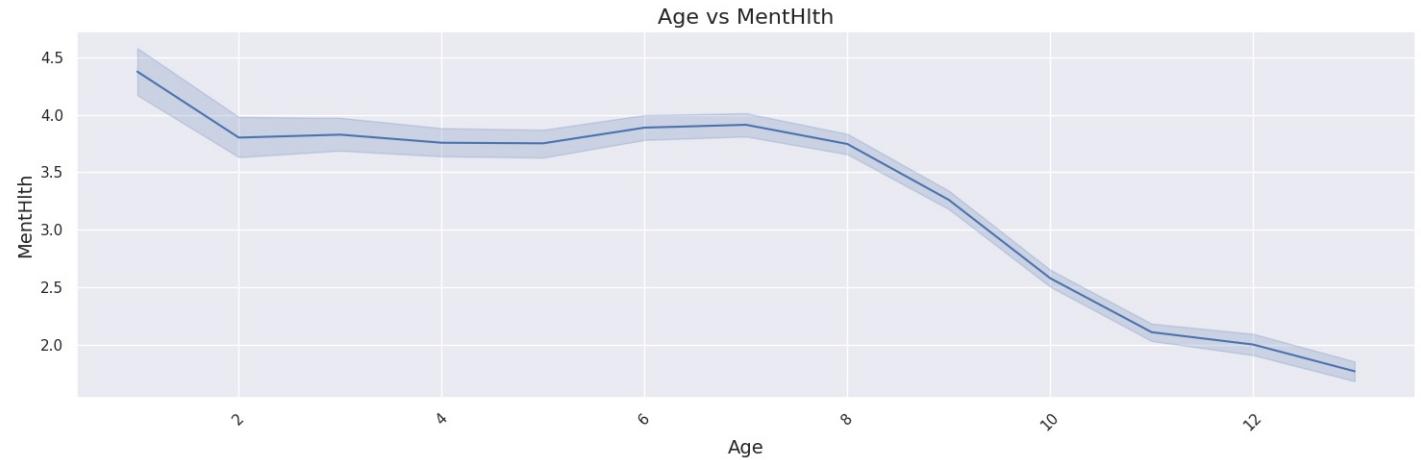
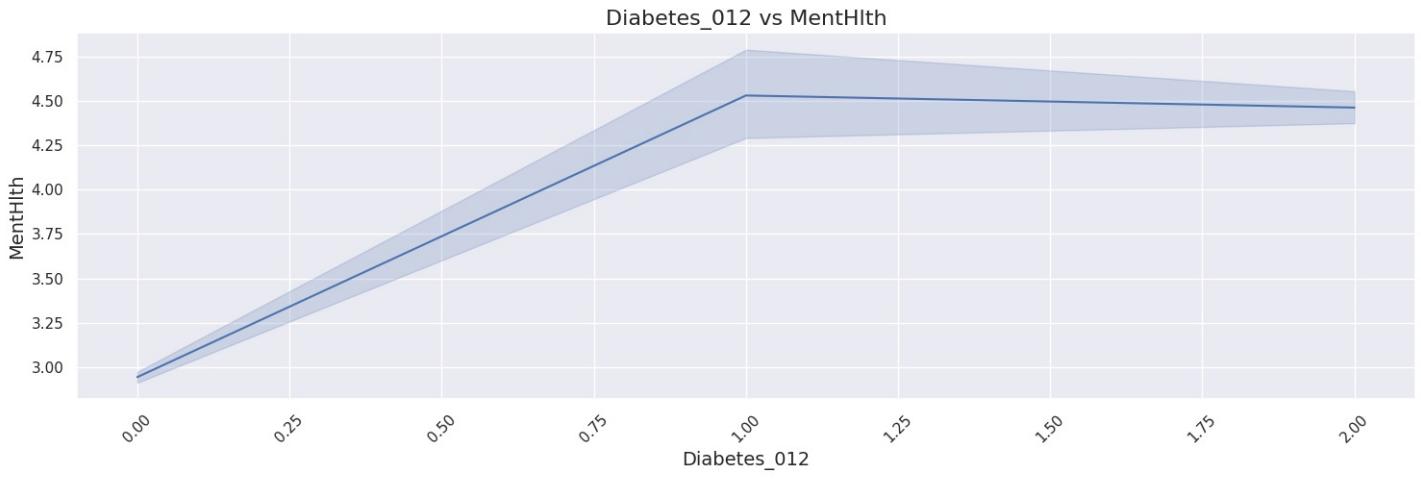
```

In [1]: # Define the numerical feature for the y axis for BMI
numerical_feature = 'MentHlt'
# Set up the matplotlib figure and size
plt.figure(figsize=(15, len(categorical) * 5))

# Iterate over each categorical feature to create a line plot
for index, feature in enumerate(categorical):
    plt.subplot(len(categorical), 1, index + 1)
    sns.lineplot(data=df, x=feature, y=numerical_feature)
    plt.title(f'{feature} vs {numerical_feature}', fontsize=16)
    plt.xlabel(feature, fontsize=14)
    plt.ylabel(numerical_feature, fontsize=14)
    plt.xticks(rotation=45) # Rotate x-axis labels by 45 degrees for readability

plt.tight_layout()
plt.show()

```



Diabetes_012 vs MentHlth: MentHlth (poor mental health days) increases with higher diabetes status, peaking at prediabetes (1) and slightly decreasing at diabetes (2).

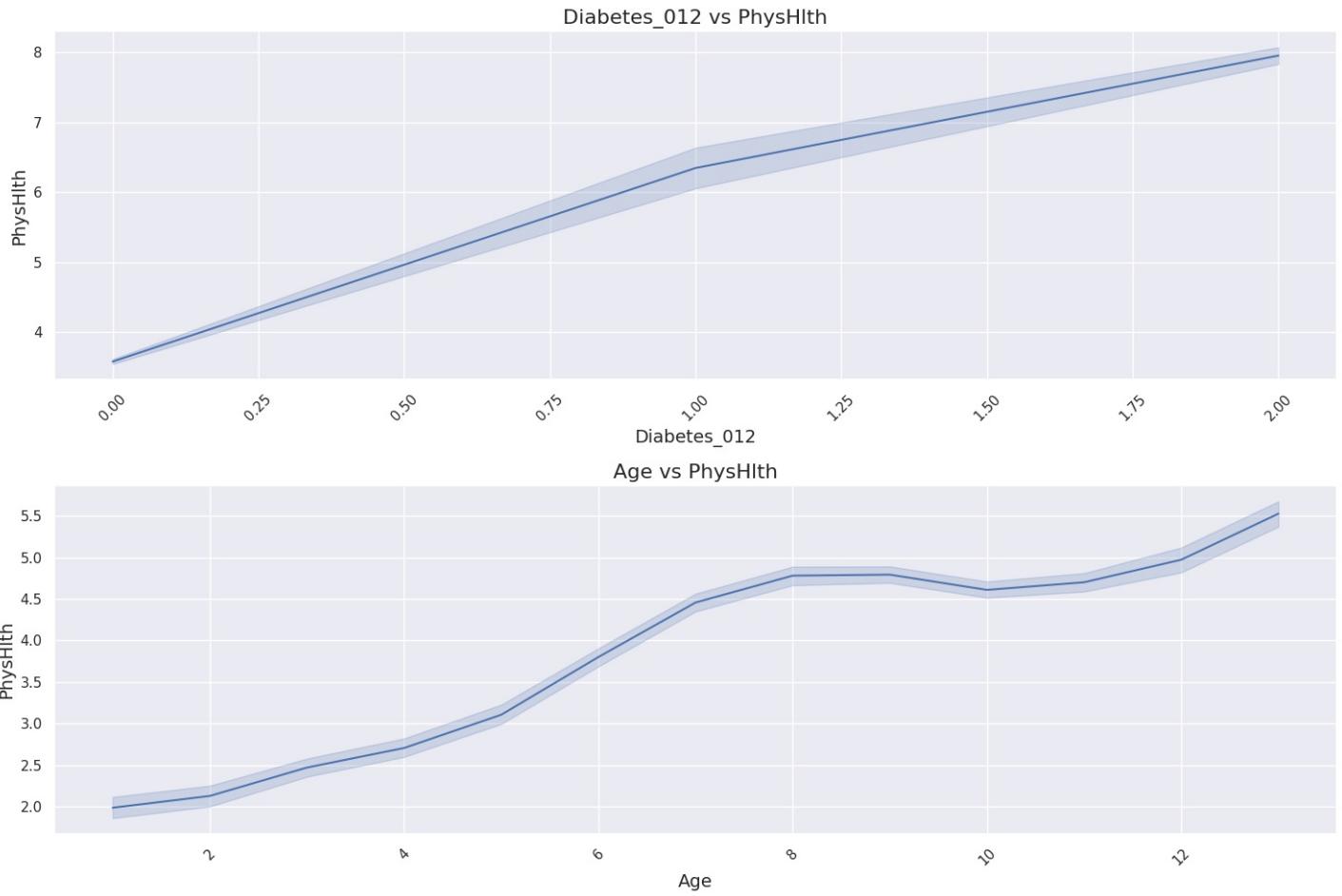
Age vs MentHlth: MentHlth decreases with age, peaking in younger age groups and gradually declining in older age groups.

Physical Health

```
# Define the numerical feature for the y axis for BMI
numerical_feature = 'PhysHlth'
# Set up the matplotlib figure and size
plt.figure(figsize=(15, len(categorical) * 5))

# Iterate over each categorical feature to create a line plot
for index, feature in enumerate(categorical):
    plt.subplot(len(categorical), 1, index + 1)
    sns.lineplot(data=df, x=feature, y=numerical_feature)
    plt.title(f'{feature} vs {numerical_feature}', fontsize=16)
    plt.xlabel(feature, fontsize=14)
    plt.ylabel(numerical_feature, fontsize=14)
    plt.xticks(rotation=45) # Rotate x-axis labels by 45 degrees for readability

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

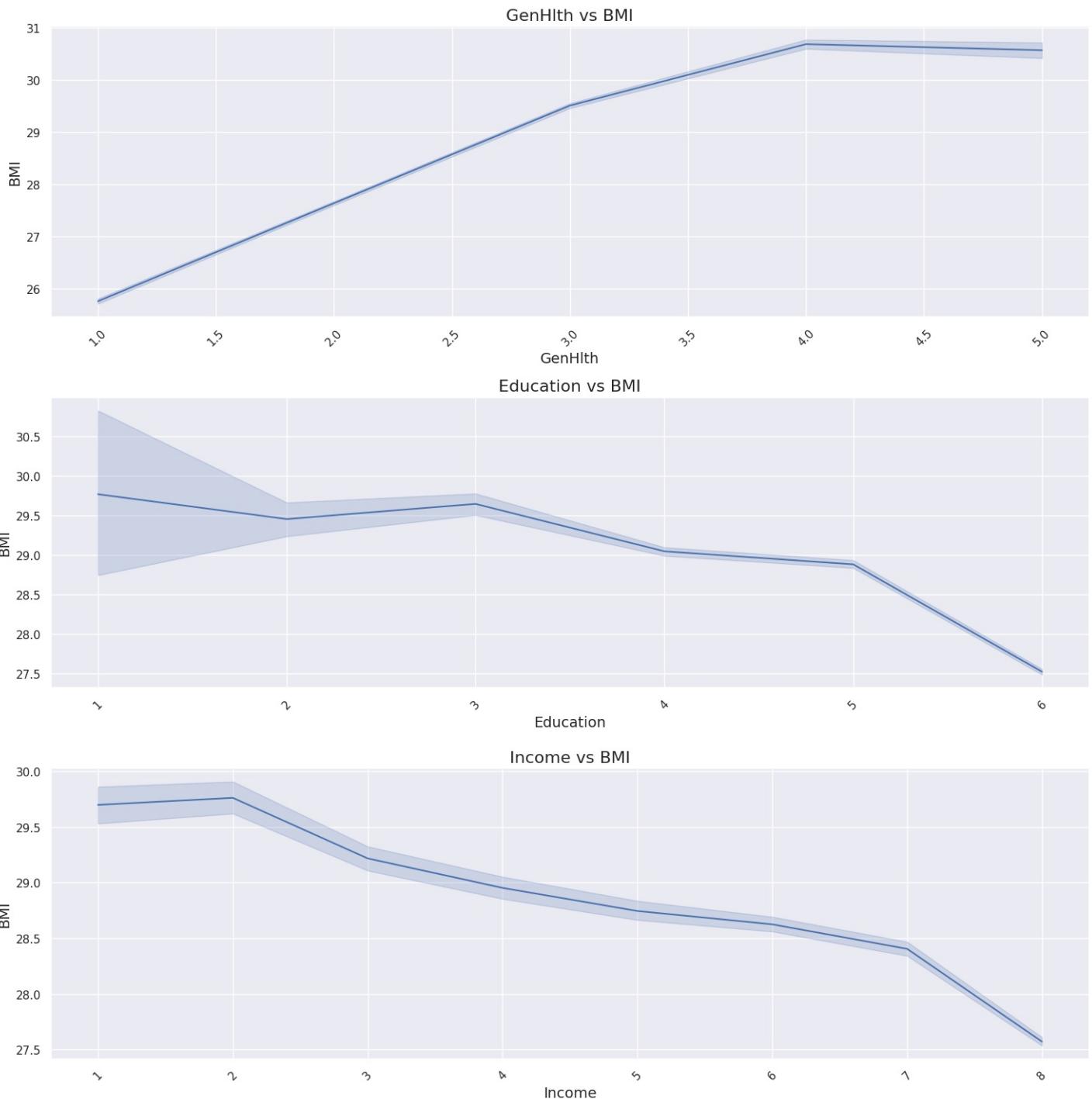


BMI vs Ordinal

```
In [ ]: # Define the numerical feature for the y axis for BMI
numerical_feature = "BMI"
# Set up the matplotlib figure and size
plt.figure(figsize=(15, len(ordinal) * 5))

# Iterate over each categorical feature to create a line plot
for index, feature in enumerate(ordinal):
    plt.subplot(len(ordinal), 1, index + 1)
    sns.lineplot(data=df, x=feature, y=numerical_feature)
    plt.title(f'{feature} vs {numerical_feature}', fontsize=16)
    plt.xlabel(feature, fontsize=14)
    plt.ylabel(numerical_feature, fontsize=14)
    plt.xticks(rotation=45) # Rotate x-axis labels by 45 degrees for readability

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



GenHlth vs BMI: BMI increases as general health worsens, from about 26 (excellent health) to 31 (poor health).

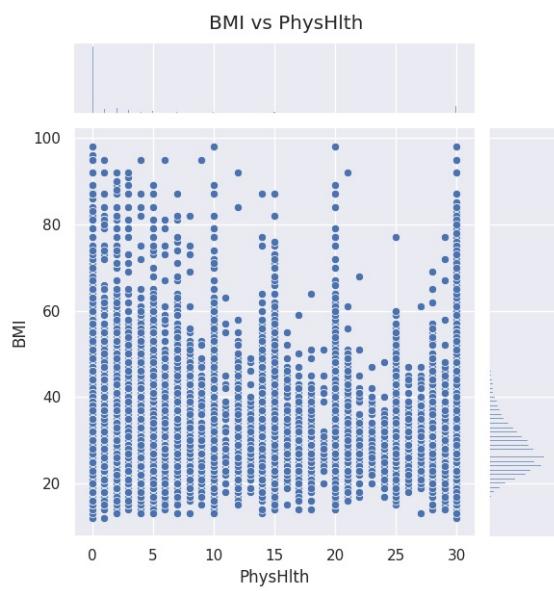
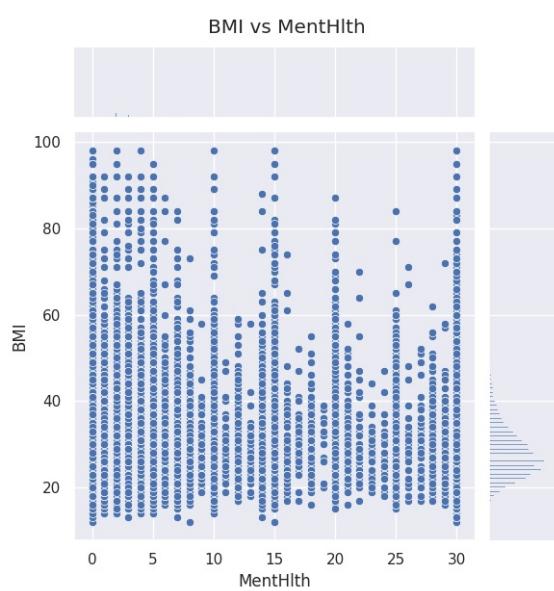
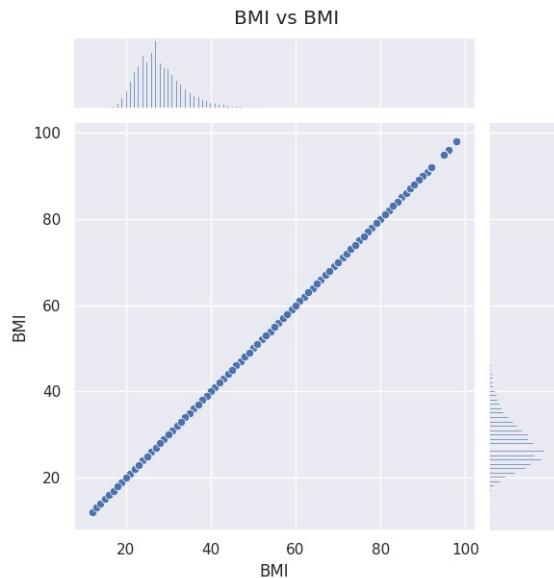
Education vs BMI: BMI decreases with higher education levels, from about 30.5 at level 1 to 27.5 at level 6.

Income vs BMI: BMI decreases with higher income levels, from about 30 at level 1 to 27.5 at level 8.

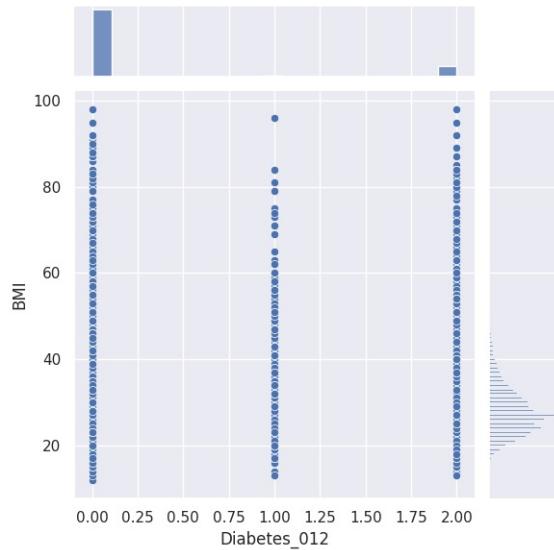
Joint Plot

```
# Features to plot against BMI
features_to_plot = numerical + categorical + binary + ordinal

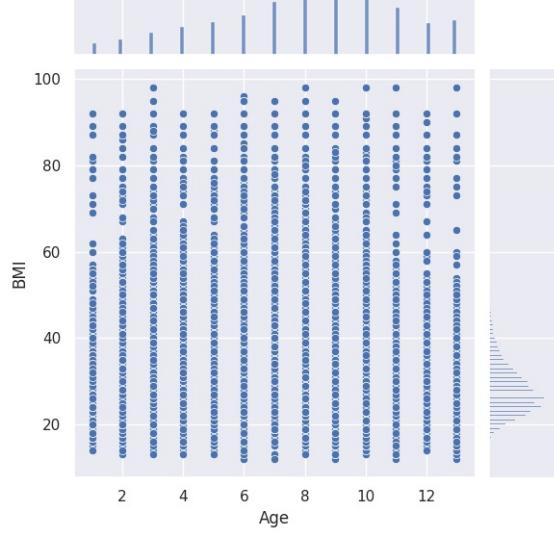
# Create joint plots for BMI against each feature
for feature in features_to_plot:
    sns.jointplot(data=df, x=feature, y='BMI', kind='scatter')
    plt.suptitle(f'BMI vs {feature}', y=1.02)
    plt.show()
```



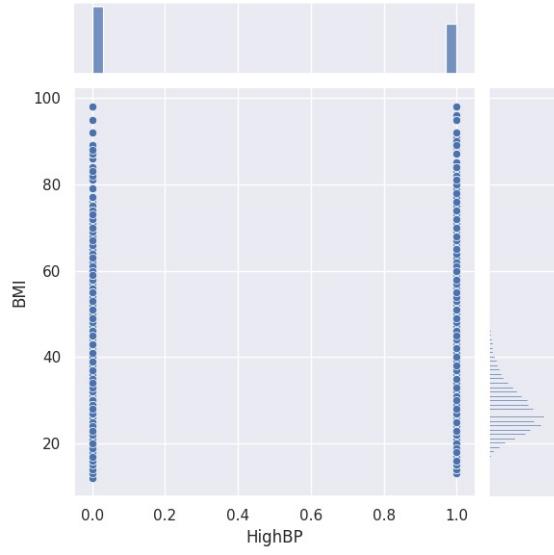
BMI vs Diabetes_012



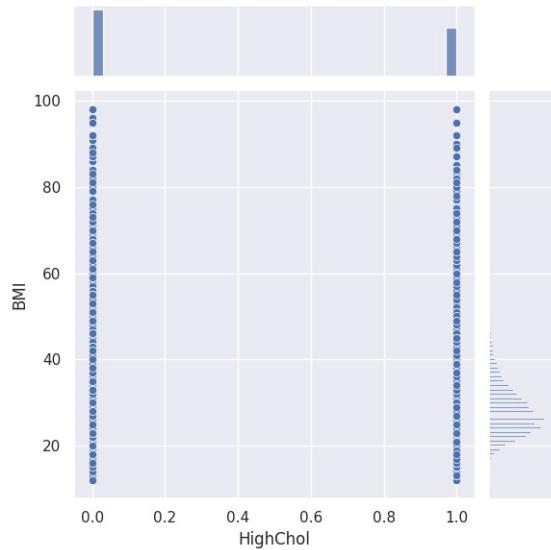
BMI vs Age



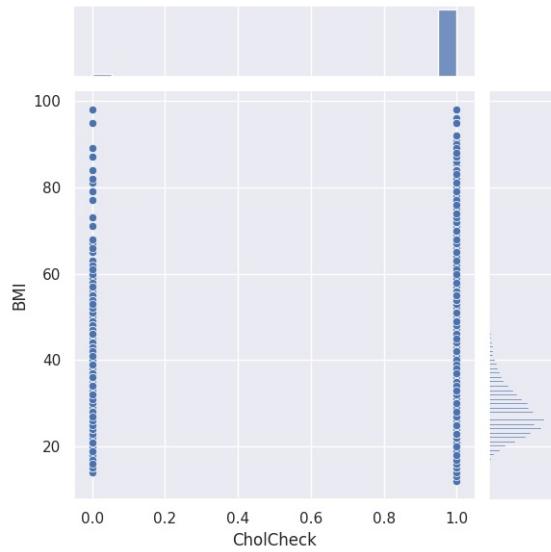
BMI vs HighBP



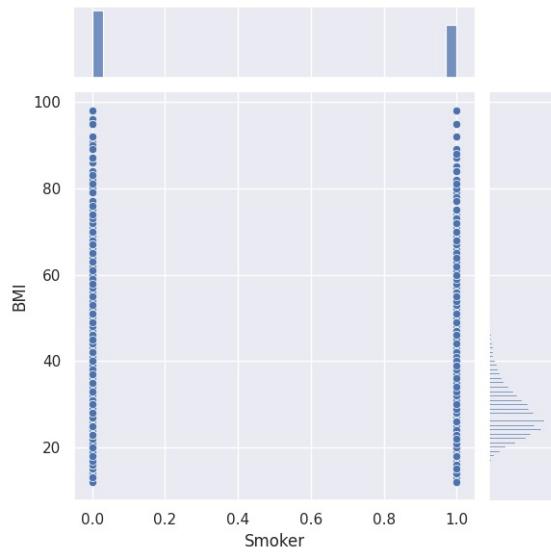
BMI vs HighChol

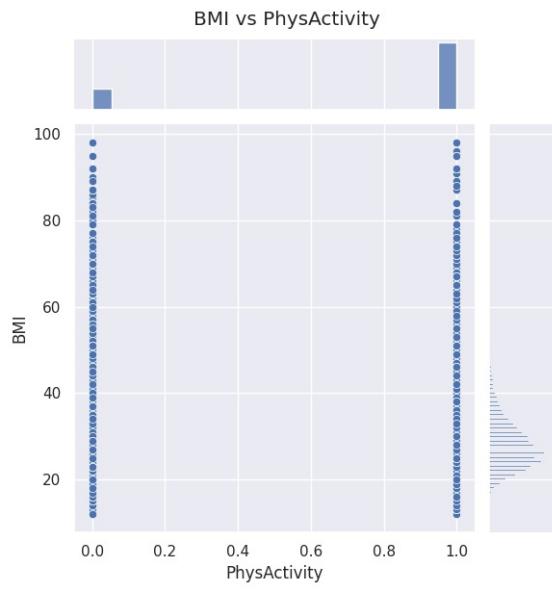
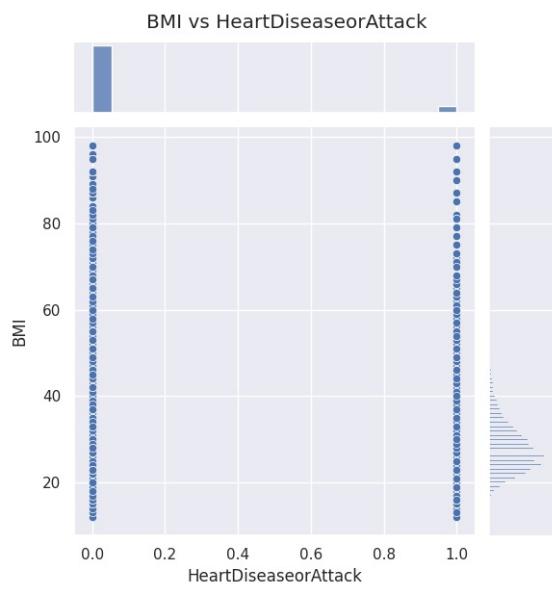
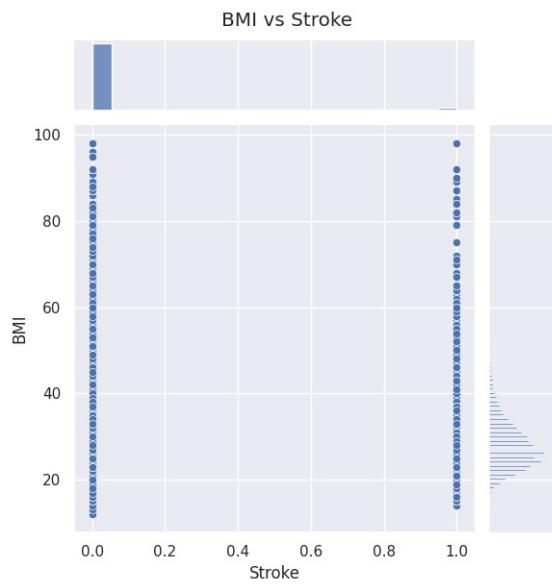


BMI vs CholCheck

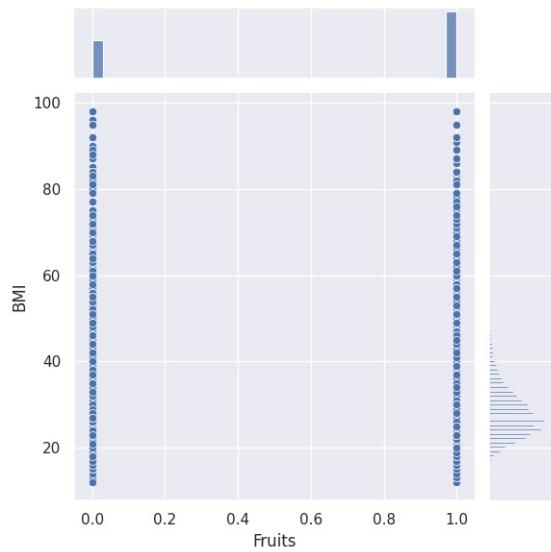


BMI vs Smoker

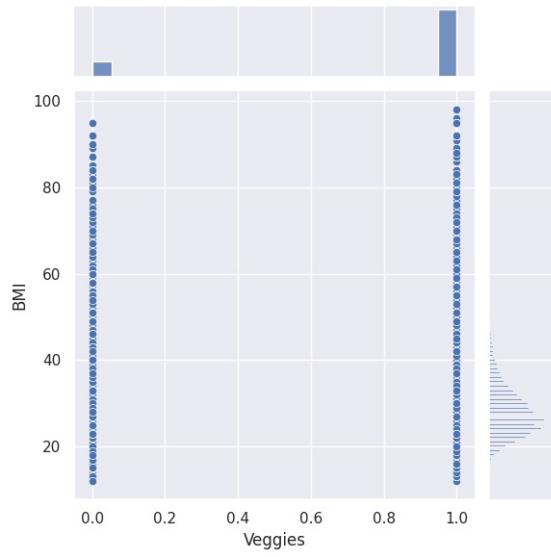




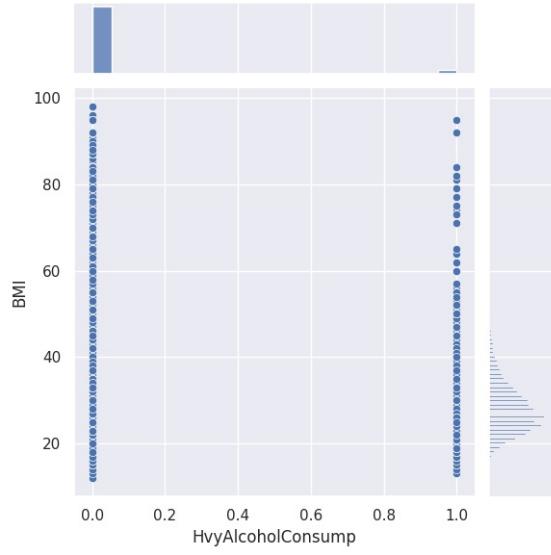
BMI vs Fruits



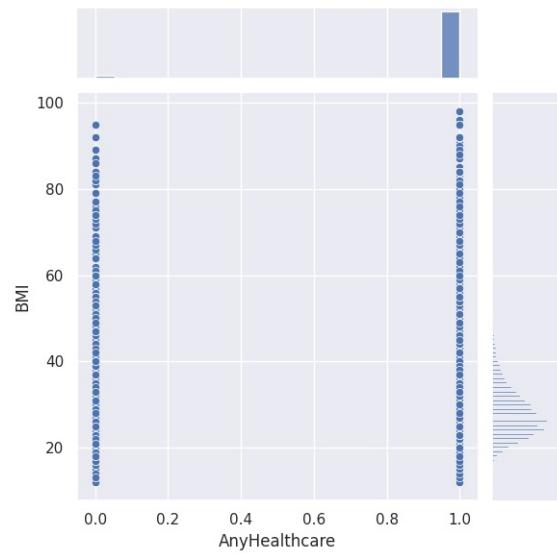
BMI vs Veggies



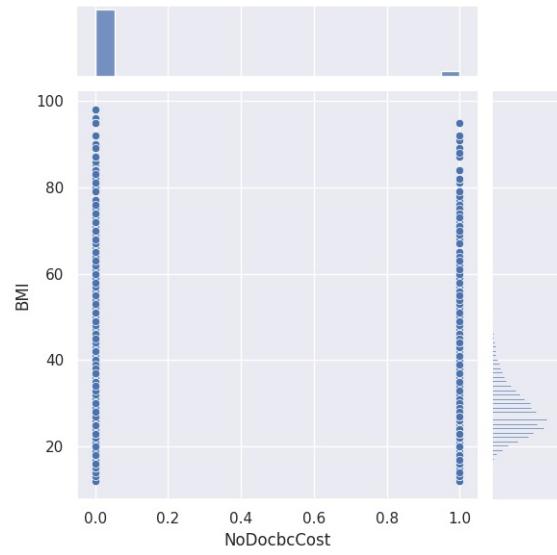
BMI vs HvyAlcoholConsump



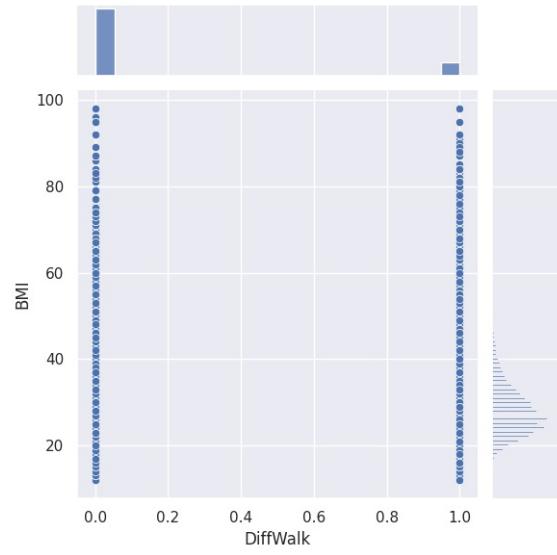
BMI vs AnyHealthcare



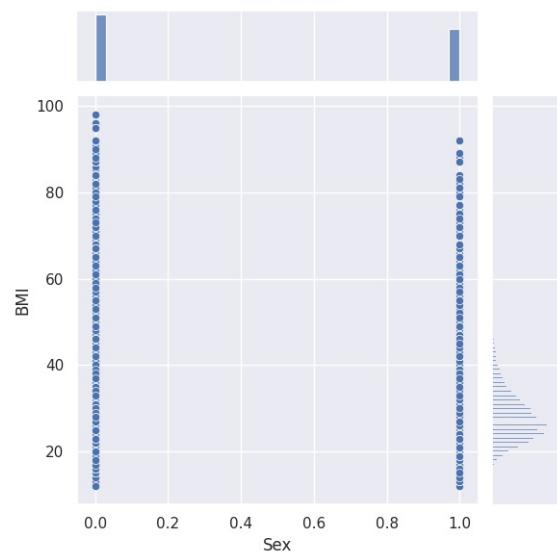
BMI vs NoDocbcCost



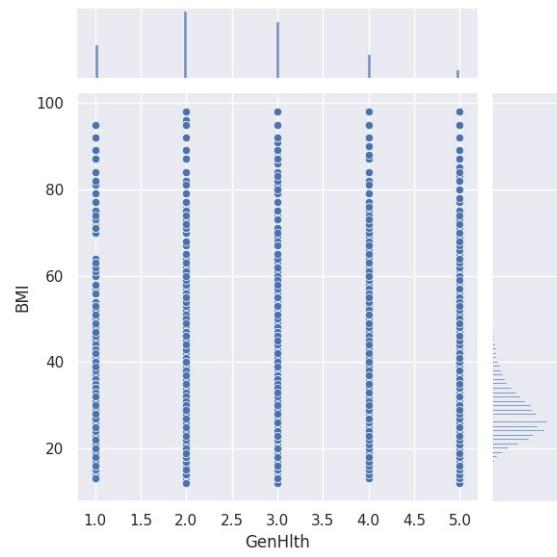
BMI vs DiffWalk



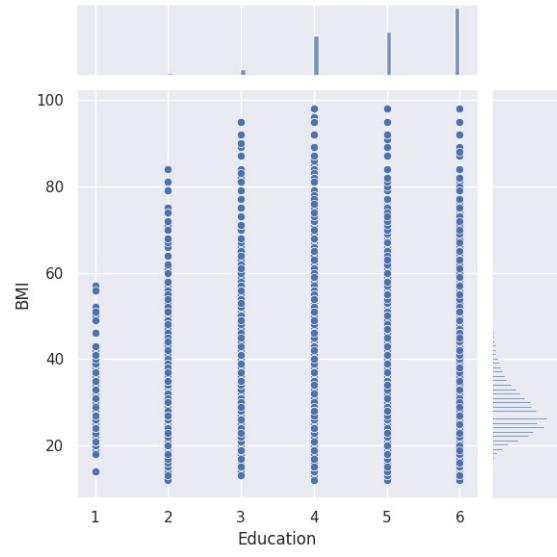
BMI vs Sex



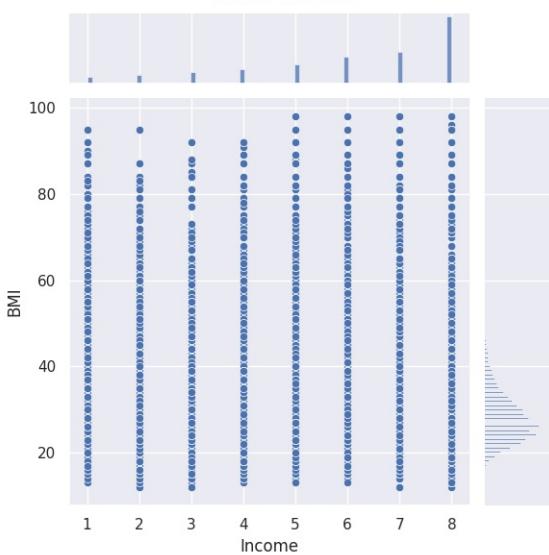
BMI vs GenHlth



BMI vs Education



BMI vs Income



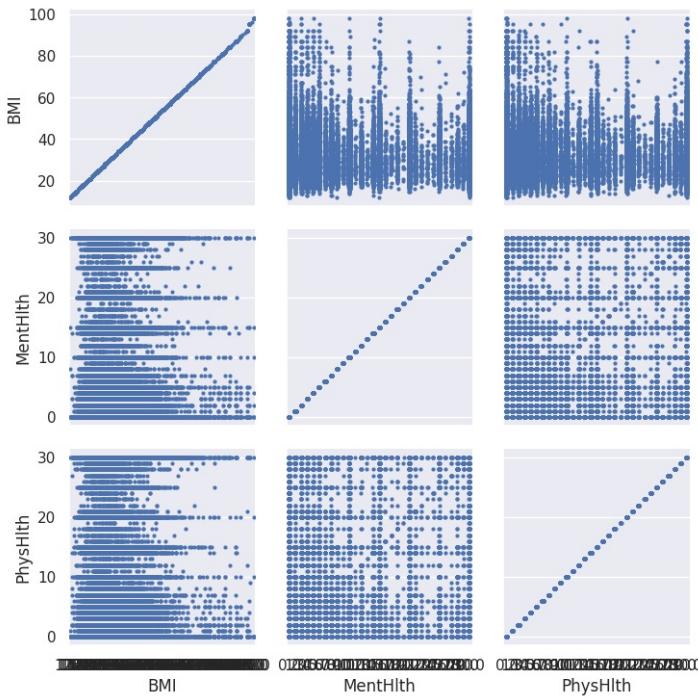
Pairgrid

```
In [1]: # Combine numerical variables for the PairGrid
pairgrid_vars = numerical

# Create the PairGrid
g = sns.PairGrid(df[pairgrid_vars])

# Map the stripplot with jitter to the PairGrid
g.map(sns.stripplot, jitter=True, size=3)

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



1. BMI:

- No clear correlation with MentHlth or PhysHlth; points are widely dispersed.

2. MentHlth:

- Scattered plots with BMI and PhysHlth, indicating no clear relationship.

3. PhysHlth:

- Scattered plots with BMI and MentHlth, indicating no clear relationship.

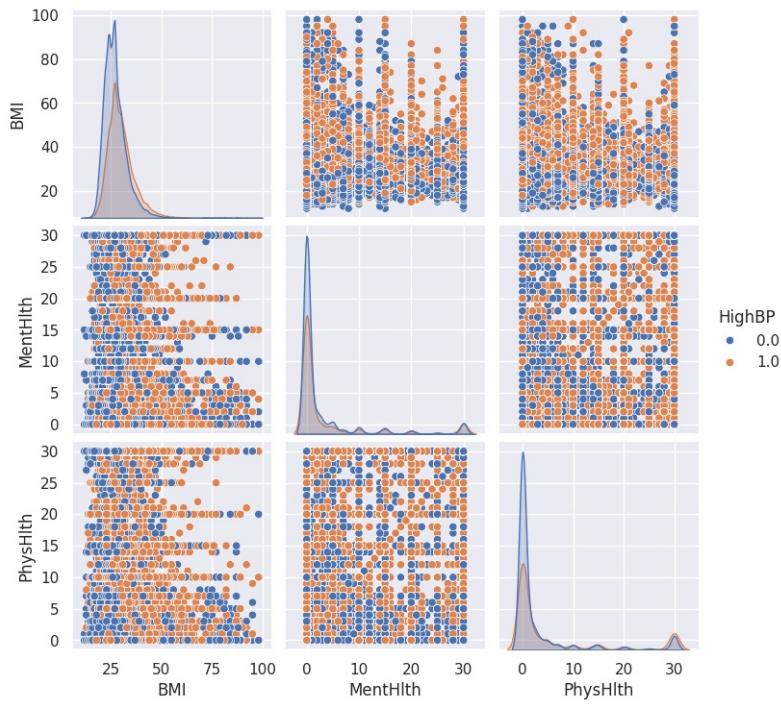
Piarpot

```
In [1]: ## Iterate through all binary features to create separate pair plots
# for bin_feature in binary:
#     # Create the pair plot with the current binary feature as hue
#     sns.pairplot(df[numerical + [bin_feature]], hue=bin_feature, diag_kind='kde')

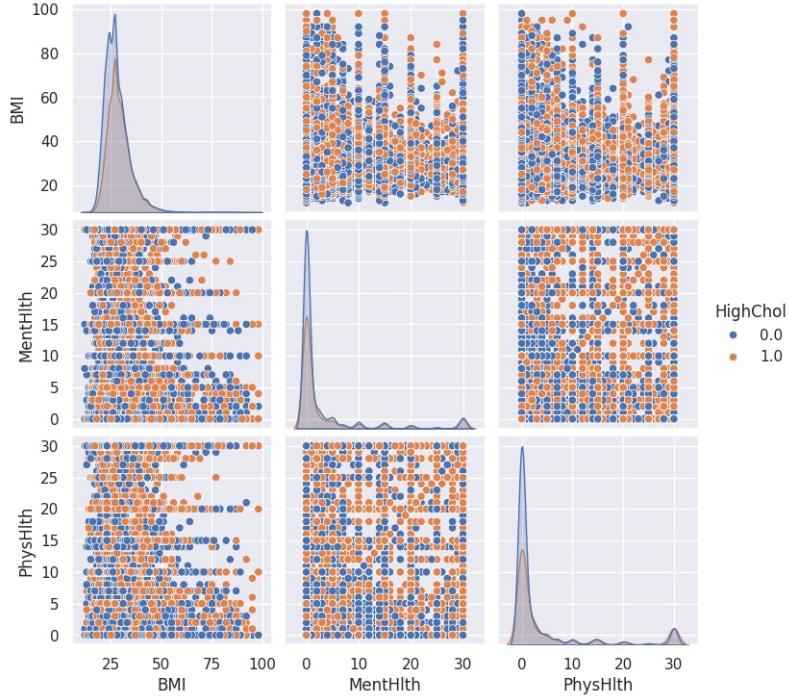
#     # Add a title to the plot
#     plt.suptitle(f'Pair Plot with {bin_feature} as Hue', y=1.02)

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

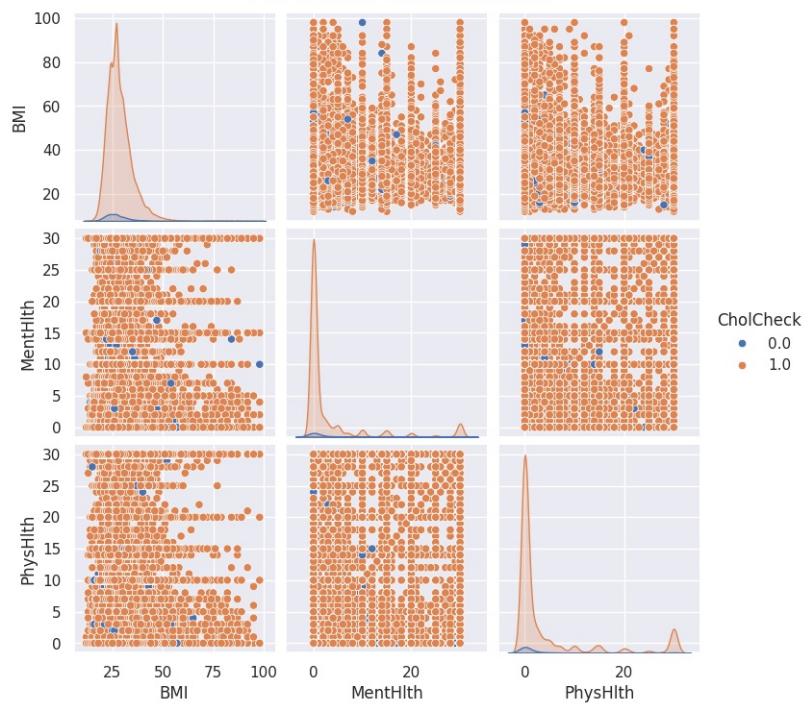
Pair Plot with HighBP as Hue



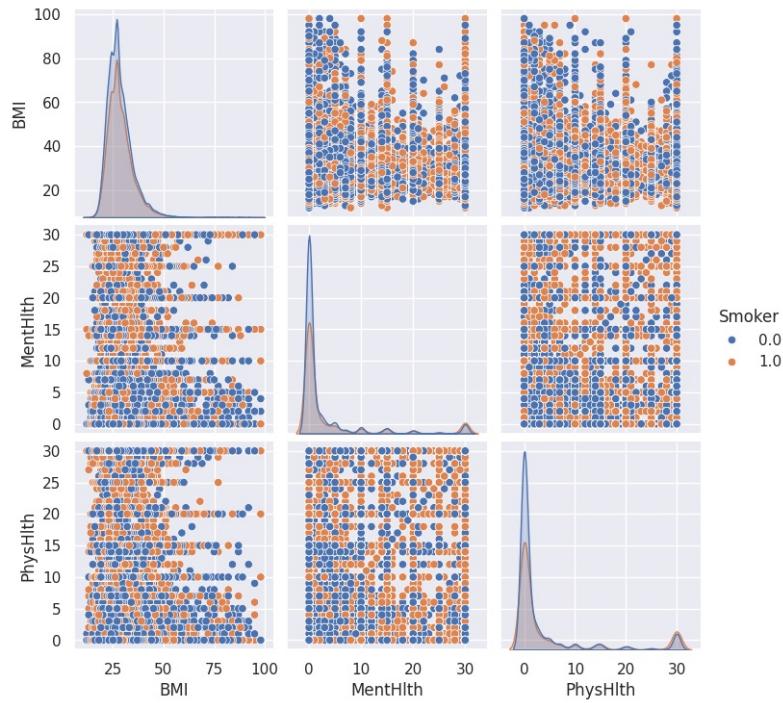
Pair Plot with HighChol as Hue



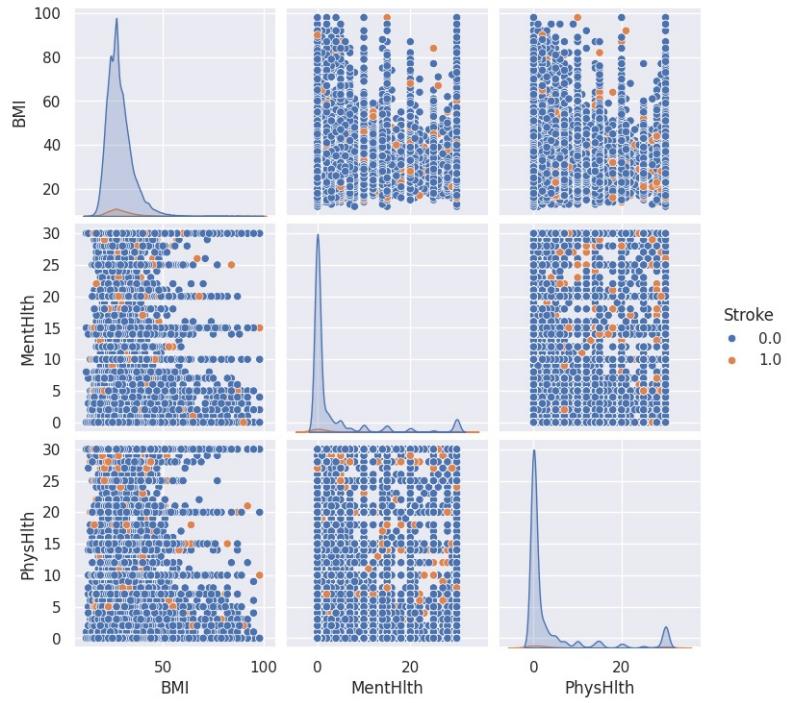
Pair Plot with CholCheck as Hue



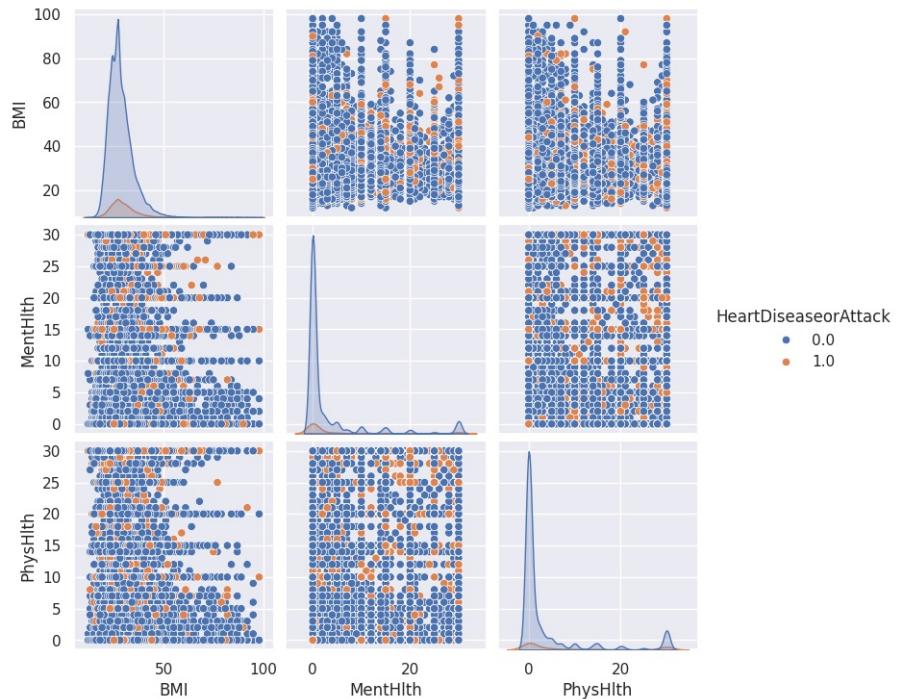
Pair Plot with Smoker as Hue



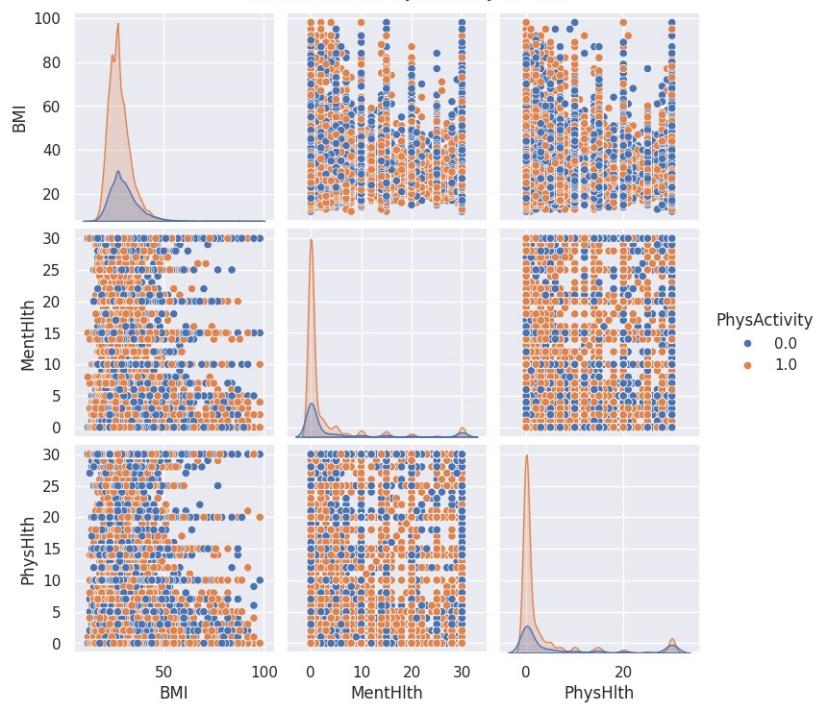
Pair Plot with Stroke as Hue



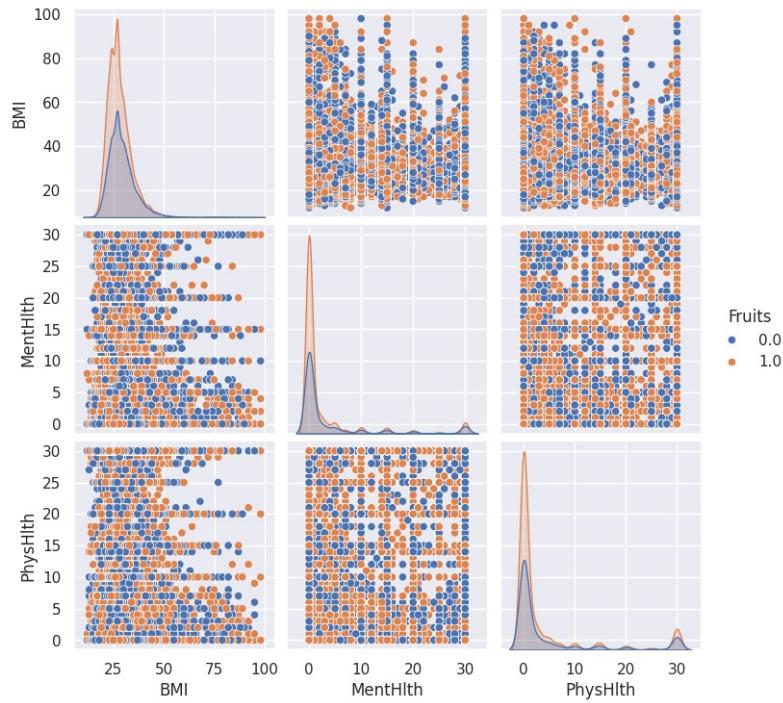
Pair Plot with HeartDiseaseorAttack as Hue



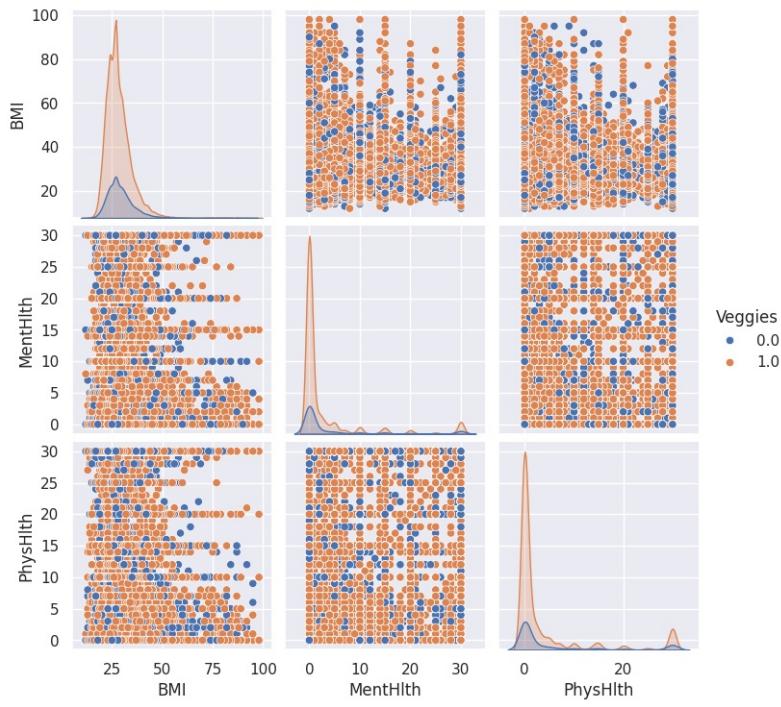
Pair Plot with PhysActivity as Hue



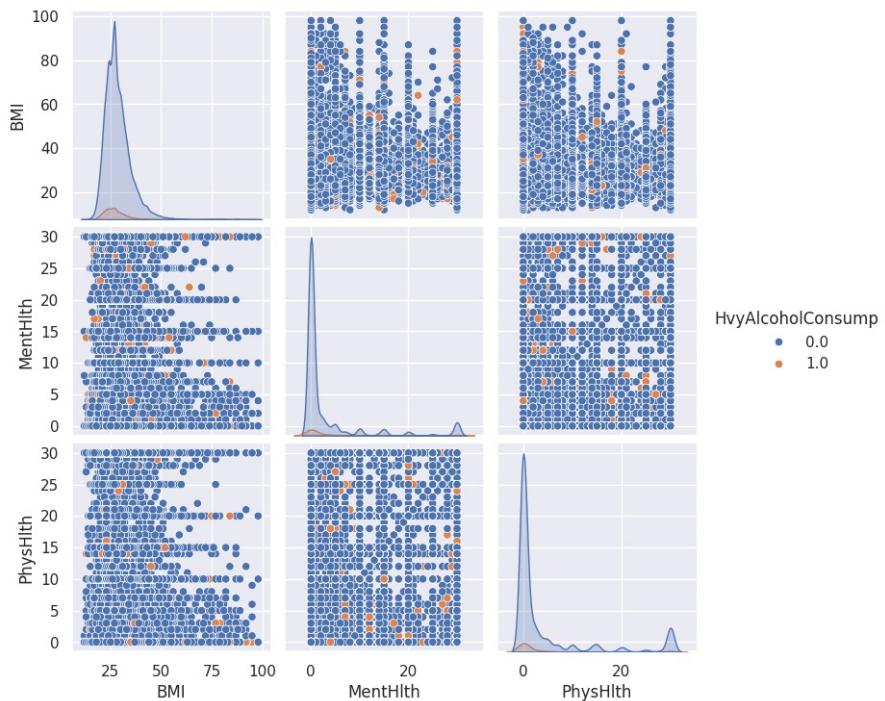
Pair Plot with Fruits as Hue



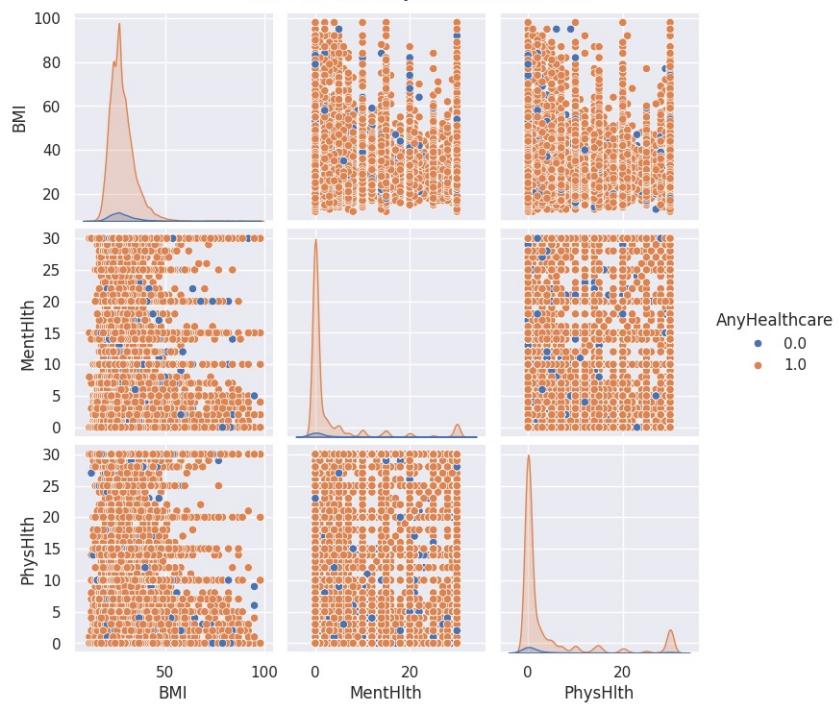
Pair Plot with Veggies as Hue



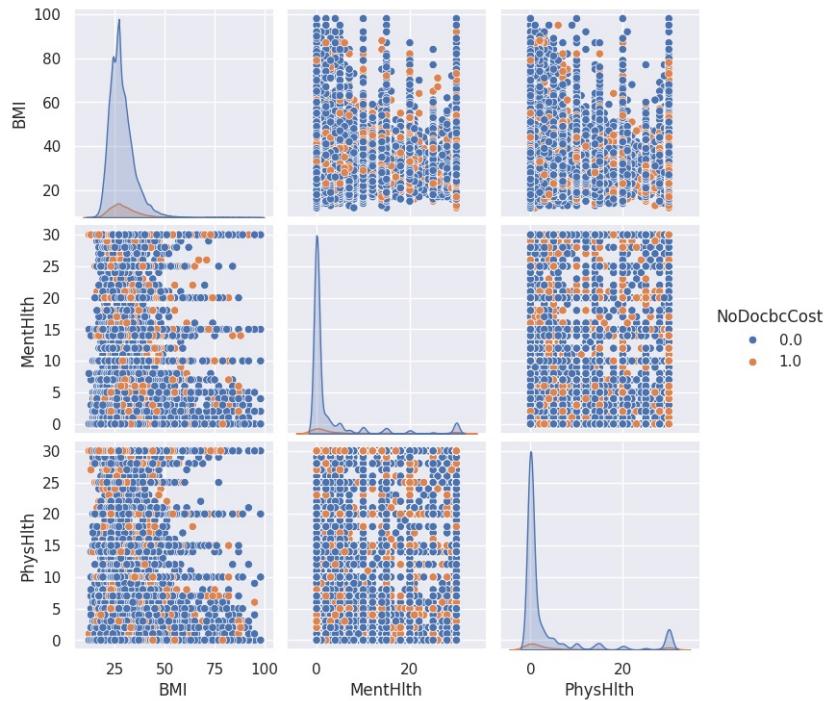
Pair Plot with HvyAlcoholConsump as Hue

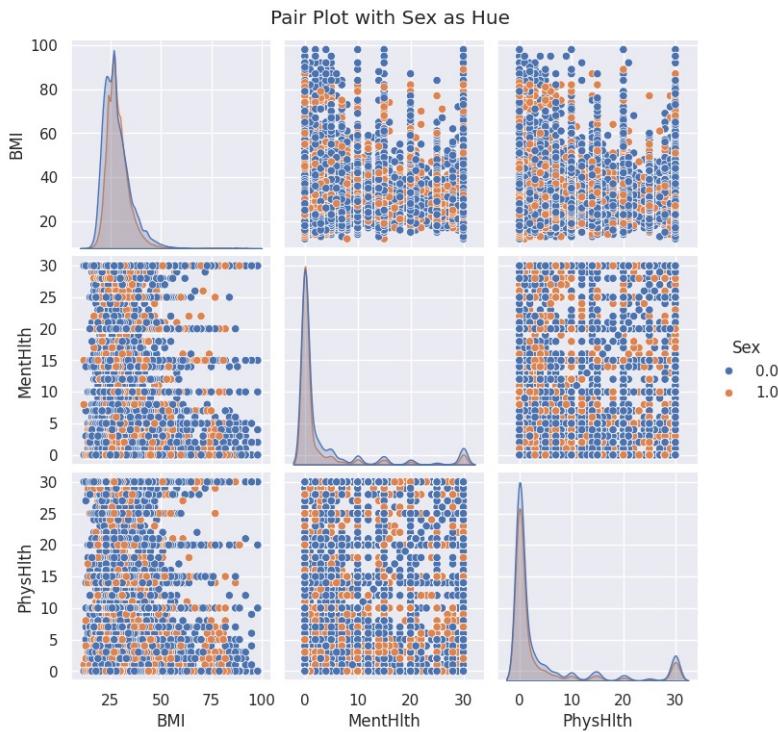
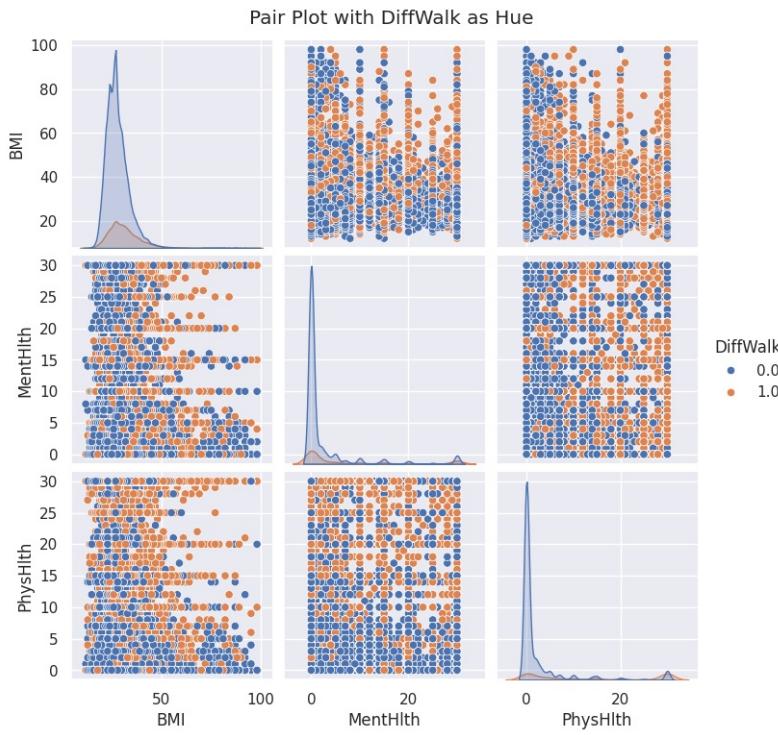


Pair Plot with AnyHealthcare as Hue



Pair Plot with NoDocbcCost as Hue





Pair Plots Summary

1. HighBP (High Blood Pressure)

- High blood pressure (orange) is more common in individuals with higher BMI.
- Distribution of MentHlth and PhysHlth shows scattered points without clear separation between high and low blood pressure groups.

2. HighChol (High Cholesterol)

- High cholesterol (orange) is associated with higher BMI.
- No clear pattern in the distribution of MentHlth and PhysHlth between individuals with and without high cholesterol.

3. CholCheck (Cholesterol Check)

- Majority of individuals have undergone cholesterol checks (orange).
- No significant difference in the distribution of BMI, MentHlth, or PhysHlth based on cholesterol check status.

4. Smoker

- Smoking status (orange) shows no clear correlation with BMI, MentHlth, or PhysHlth.
- Both smokers and non-smokers are scattered similarly across the plots.

5. Stroke

- Stroke occurrence (orange) is relatively rare and shows no clear pattern with BMI, MentHlth, or PhysHlth.
- Individuals with strokes are scattered throughout the plots.

6. HeartDiseaseorAttack

- Heart disease or attack (orange) is associated with higher BMI.
- No clear separation in the distribution of MentHlth and PhysHlth based on heart disease status.

7. PhysActivity (Physical Activity)

- Physically active individuals (orange) are spread across various BMI, MentHlth, and PhysHlth values.
- No significant difference in distribution patterns for physical activity.

8. Fruits

- Fruit consumption (orange) shows a slight association with lower BMI.
- No clear separation in MentHlth and PhysHlth distributions based on fruit consumption.

9. Veggies

- Vegetable consumption (orange) also shows a slight association with lower BMI.
- Distribution of MentHlth and PhysHlth is similar across those who consume vegetables and those who do not.

10. HvyAlcoholConsump (Heavy Alcohol Consumption)

- Heavy alcohol consumption (orange) shows no clear pattern with BMI, MentHlth, or PhysHlth.
- Both heavy drinkers and non-drinkers are similarly scattered across the plots.

11. AnyHealthcare

- Most individuals have access to healthcare (orange).
- No significant difference in BMI, MentHlth, or PhysHlth distributions based on healthcare access.

12. NoDocbcCost (No Doctor because of Cost)

- Individuals who could not see a doctor due to cost (orange) show no clear pattern in BMI, MentHlth, or PhysHlth.
- Both groups are similarly scattered across the plots.

13. DiffWalk (Difficulty Walking)

- Individuals with difficulty walking (orange) have higher BMIs compared to those without difficulty walking.
- No clear separation in the distribution of MentHlth based on difficulty walking.
- Those with difficulty walking have more poor physical health days (PhysHlth) compared to those without difficulty walking.

Feature Engineering

```
In [14]: # Handle missing values for numerical variables
df[numerical] = df[numerical].fillna(df[numerical].median())

# Encode ordinal variables
le = LabelEncoder()
for col in ordinal:
    df[col] = le.fit_transform(df[col])

# Define the target and features
target_column = 'Diabetes_012'
features = df.columns.difference([target_column])
X = df[features]
y = df[target_column]

In [15]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the numerical features
scaler = StandardScaler()
X_train[numerical] = scaler.fit_transform(X_train[numerical])
X_test[numerical] = scaler.transform(X_test[numerical])

# Generate oversample with SMOTE:
smote = SMOTE(random_state=42)
X_oversampled, y_oversampled = smote.fit_resample(X_train, y_train)

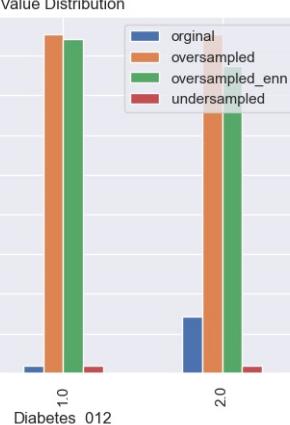
# Generate oversample with SMOTENN:
smotenn = SMOTENN(random_state=42)
X_oversampled_enn, y_oversampled_enn = smotenn.fit_resample(X_train, y_train)

# Generate undersample with RandomUnderSampler:
undersampler = RandomUnderSampler(random_state=42)
X_undersampled, y_undersampled = undersampler.fit_resample(X_train, y_train)

values_distribution = [
    'original': y_train.value_counts(),
    'oversampled': y_oversampled.value_counts(),
    'oversampled_enn': y_oversampled_enn.value_counts(),
    'undersampled': y_undersampled.value_counts()
]

# plot value distribution
plt.figure(figsize=(16, 8))
pd.DataFrame(values_distribution).plot(kind='bar')
plt.title('Value Distribution')
plt.show()
```

<Figure size 1600x800 with 0 Axes>



Feature Selection

Chi-Square Test

```
In [17]: # DataFrame to store result of Chi2 test
results_df_cat = pd.DataFrame(columns=['Feature', 'Chi2', 'P-Value'])

# Copy to keep the original dataframe
cat_df = df.copy()

# for all non-numerical variables, convert to categorical
for var in non_numerical:
    cat_df[var] = cat_df[var].astype('category')
    print(f'Variable {var} converted to categorical')

# For each categorical variable perform Chi2 test
# Print results, create bar plot
for var in non_numerical:
    contingency_table = pd.crosstab(cat_df[var], cat_df['Diabetes_012'])
    chi2, p, __, __ = chi2_contingency(contingency_table)
```

```

print("Chi-Squared Test for [var] and Diabetes_012")
print("Chi2 value = [chi2], p-value = [p]\n")
# Add the results to the DataFrame
results_df_cat = pd.concat([results_df_cat, pd.DataFrame({'Feature': [var], 'Chi2': [chi2], 'P-Value': [p]}), ignore_index=True])

# Display the results
print(results_df_cat)

# Plotting the Chi2 values for each feature
plt.figure(figsize=(12, 8))
plt.bar(results_df_cat['Feature'], results_df_cat['Chi2'])
plt.xlabel('Features')
plt.ylabel('Chi2 Value')
plt.title('Chi2 Value for Each Categorical Feature')
plt.xticks(rotation=90)
plt.show()

Variable Diabetes_012 converted to categorical
Variable Age converted to categorical
Variable HighBP converted to categorical
Variable HighChol converted to categorical
Variable CholCheck converted to categorical
Variable Smoker converted to categorical
Variable Stroke converted to categorical
Variable HeartDiseaseorAttack converted to categorical
Variable PhysActivity converted to categorical
Variable Fruits converted to categorical
Variable Veggies converted to categorical
Variable HvyAlcoholConsump converted to categorical
Variable AnyHealthcare converted to categorical
Variable NoDocbcCost converted to categorical
Variable DiffWalk converted to categorical
Variable Sex converted to categorical
Variable GenHlth converted to categorical
Variable Education converted to categorical
Variable Income converted to categorical
Chi-Squared Test for Diabetes_012 and Diabetes_012
Chi2 value = 507360.0, p-value = 0.0

Chi-Squared Test for Age and Diabetes_012
Chi2 value = 9641.376530679845, p-value = 0.0

Chi-Squared Test for HighBP and Diabetes_012
Chi2 value = 18794.644052016425, p-value = 0.0

Chi-Squared Test for HighChol and Diabetes_012
Chi2 value = 11258.920399414841, p-value = 0.0

Chi-Squared Test for CholCheck and Diabetes_012
Chi2 value = 1173.749357770035, p-value = 1.3291236675197173e-255

Chi-Squared Test for Smoker and Diabetes_012
Chi2 value = 1010.5117511111928, p-value = 3.7167324294119075e-220

Chi-Squared Test for Stroke and Diabetes_012
Chi2 value = 2916.75197962113, p-value = 0.0

Chi-Squared Test for HeartDiseaseorAttack and Diabetes_012
Chi2 value = 8244.88910662167, p-value = 0.0

Chi-Squared Test for PhysActivity and Diabetes_012
Chi2 value = 3789.3014625427313, p-value = 0.0

Chi-Squared Test for Fruits and Diabetes_012
Chi2 value = 454.3470587241542, p-value = 2.1867028126650155e-99

Chi-Squared Test for Veggies and Diabetes_012
Chi2 value = 893.8419053866104, p-value = 8.029645985781328e-195

Chi-Squared Test for HvyAlcoholConsump and Diabetes_012
Chi2 value = 850.3240478355594, p-value = 2.2619296719502035e-185

Chi-Squared Test for AnyHealthcare and Diabetes_012
Chi2 value = 69.07797672213422, p-value = 9.997880563068128e-16

Chi-Squared Test for NoDocbcCost and Diabetes_012
Chi2 value = 396.08182159008913, p-value = 9.81579822340756e-87

Chi-Squared Test for DiffWalk and Diabetes_012
Chi2 value = 12776.94188915485, p-value = 0.0

Chi-Squared Test for Sex and Diabetes_012
Chi2 value = 250.85057509520166, p-value = 3.376678611575899e-55

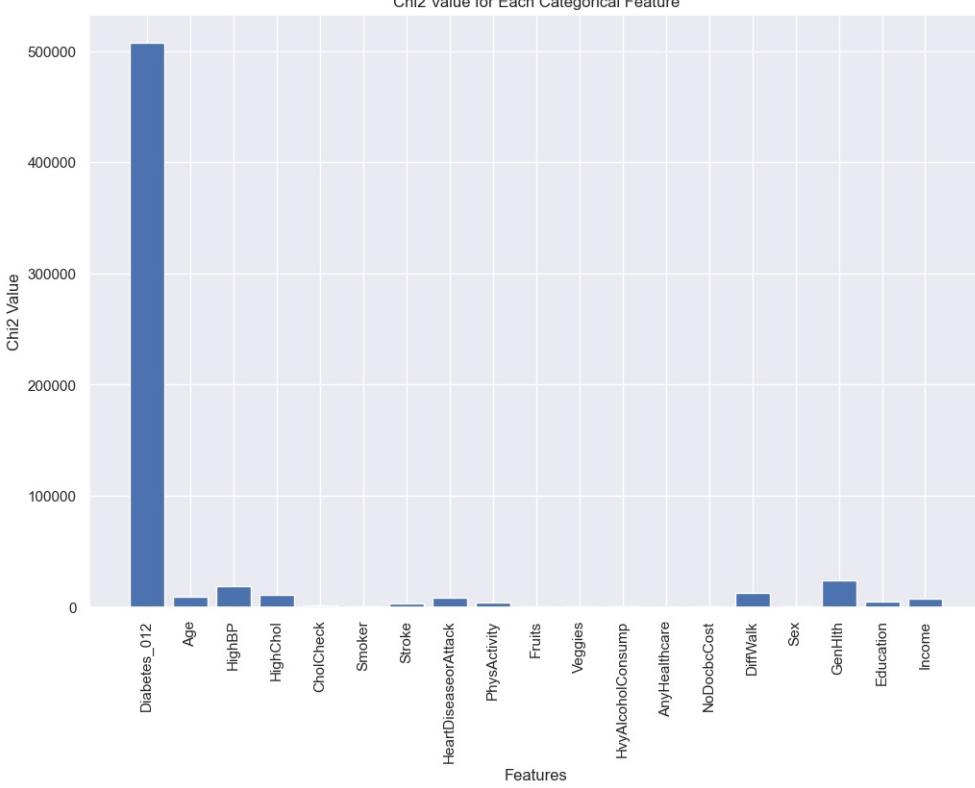
Chi-Squared Test for GenHlth and Diabetes_012
Chi2 value = 24248.10614736849, p-value = 0.0

Chi-Squared Test for Education and Diabetes_012
Chi2 value = 4560.6402794568585, p-value = 0.0

Chi-Squared Test for Income and Diabetes_012
Chi2 value = 7816.462905911266, p-value = 0.0

    Feature      Chi2      P-Value
0   Diabetes_012  507360.000000  0.000000e+00
1       Age     9641.376531  0.000000e+00
2     HighBP    18794.644052  0.000000e+00
3     HighChol   11258.920399  0.000000e+00
4     CholCheck   1173.749358  1.329124e-255
5     Smoker    1010.511751  3.716732e-220
6     Stroke    2916.751980  0.000000e+00
7  HeartDiseaseorAttack  8244.889107  0.000000e+00
8     PhysActivity  3789.301463  0.000000e+00
9       Veggies    454.347059  2.186703e-99
10  HvyAlcoholConsump  893.841905  8.029646e-195
11  AnyHealthcare    69.077977  9.997881e-185
12  NoDocbcCost    396.081822  9.815790e-87
13  DiffWalk    12776.941889  0.000000e+00
14       Sex     250.850575  3.376679e-55
15  GenHlth    24248.106147  0.000000e+00
16  Education    4560.640279  0.000000e+00
17    Income    7816.462906  0.000000e+00

```



```
In [18]: # List all the values with p-value less than 0.05
significant_cat = results_df_cat[results_df_cat['P-Value'] < 0.05]
display(significant_cat)
```

	Feature	Chi2	P-Value
0	Diabetes_012	507360.000000	0.000000e+00
1	Age	9641.376531	0.000000e+00
2	HighBP	18794.644052	0.000000e+00
3	HighChol	11258.920399	0.000000e+00
4	CholCheck	1173.749358	1.329124e-255
5	Smoker	1010.511751	3.716732e-220
6	Stroke	2916.751980	0.000000e+00
7	HeartDiseaseorAttack	8244.889107	0.000000e+00
8	PhysActivity	3789.301463	0.000000e+00
9	Fruits	454.347059	2.186703e-99
10	Veggies	893.841905	8.029646e-195
11	HvyAlcoholConsump	850.324048	2.261930e-185
12	AnyHealthcare	69.077977	9.997881e-16
13	NoDocbcCost	396.081822	9.815790e-87
14	DiffWalk	12776.941889	0.000000e+00
15	Sex	250.850575	3.376679e-55
16	GenHlth	24248.106147	0.000000e+00
17	Education	4560.640279	0.000000e+00
18	Income	7816.462906	0.000000e+00

T-Test

```
In [20]: results_df_nums = pd.DataFrame(columns=['Feature', 'Statistic', 'P-Value'])

for var in numerical:
    # List to hold test results for each category vs. rest
    for category in df[target_column].unique():
        group1 = df[df[target_column] == category][var]
        group2 = df[df[target_column] != category][var]
        stat, p = ttest_ind(group1, group2)

    # Add the results to the DataFrame
    results_df_nums = pd.concat([results_df_nums, pd.DataFrame({'Feature': [var], 'Statistic': [stat], 'P-Value': [p]}), ignore_index=True])

# Display the results
display(results_df_nums)
```

	Feature	Statistic	P-Value
0	BMI	-115.681258	0.000000e+00
1	BMI	111.878142	0.000000e+00
2	BMI	24.368855	5.183573e-131
3	MentHlth	-37.866752	0.000000e+00
4	MentHlth	34.995722	1.143283e-267
5	MentHlth	12.466688	1.162108e-35
6	PhysHlth	-89.495537	0.000000e+00
7	PhysHlth	87.591482	0.000000e+00
8	PhysHlth	16.602108	7.255186e-62

```
In [21]: # List all the values with p-value less than 0.05
significant_nums = results_df_nums[results_df_nums['P-Value'] < 0.05]
display(significant_nums)
```

Feature	Statistic	P-Value
0 BMI	-115.681258	0.000000e+00
1 BMI	111.878142	0.000000e+00
2 BMI	24.368855	5.183573e-131
3 MentHlth	-37.866752	0.000000e+00
4 MentHlth	34.995722	1.143283e-267
5 MentHlth	12.466688	1.162108e-35
6 PhysHlth	-89.495537	0.000000e+00
7 PhysHlth	87.591482	0.000000e+00
8 PhysHlth	16.602108	7.255186e-62

```
In [22]: # compare length of significant variables
print("Number of total categorical variables: " + str(len(non_numerical)))
print("Number of significant categorical variables: " + str(len(significant_cat)))
print("Number of total numerical variables: " + str(len(numerical)))
print("Number of significant numerical variables: " + str(len(significant_nums)))
```

Number of total categorical variables: 19
Number of significant categorical variables: 19
Number of total numerical variables: 3
Number of significant numerical variables: 9

Conclusion for Significance tests

Chi-Square Test Results

- Age, HighBP, HighChol, Stroke, HeartDiseaseorAttack, PhysActivity, DiffWalk, GenHlth, Education, Income: These variables also show very high chi-square values and p-values of 0.000000e+00, suggesting strong associations with diabetes.
- CholCheck, Smoker, Fruits, Veggies, HvyAlcoholConsump, AnyHealthcare, NoDocbcCost, Sex: These variables have lower chi-square values compared to the above but still show significant associations with diabetes, as indicated by their p-values.

T-Test Result

- BMI: The t-values (both positive and negative) and p-values (0.000000e+00) indicate significant differences in BMI between diabetic and non-diabetic groups.
- MentHlth: The t-values (both positive and negative) and extremely low p-values suggest significant differences in mental health scores between the groups.
- PhysHlth: Similar to BMI and MentHlth, the t-values and p-values indicate significant differences in physical health scores between diabetic and non-diabetic groups.

Interpretation

- Chi-Square Test: Variables like Age, HighBP, HighChol, Stroke, HeartDiseaseorAttack, PhysActivity, DiffWalk, GenHlth, Education, and Income have *strong associations with diabetes*.
- T-Test: Variables like BMI, MentHlth, and PhysHlth show significant differences in their means between diabetic and non-diabetic groups, indicating these factors are important in predicting diabetes.

Modeling

Helper Functions

```
In [23]: def train_and_evaluate_svm(X_train, X_test, y_train, y_test, random_state=42):
    svm_classifier = SVC(random_state=random_state, probability=True)
    svm_classifier.fit(X_train, y_train)
    y_pred = svm_classifier.predict(X_test)
    y_prob = svm_classifier.predict_proba(X_test)

    report_df = pd.DataFrame(classification_report(y_test, y_pred, output_dict=True)).transpose()
    auc_score = roc_auc_score(y_test, y_prob, multi_class='ovr')

    return report_df, auc_score, y_prob
```

```
In [24]: from sklearn.naive_bayes import GaussianNB

def train_and_evaluate_nb(X_train, X_test, y_train, y_test):
    # Create and train the GaussianNB classifier
    nb_classifier = GaussianNB()
    nb_classifier.fit(X_train, y_train)

    # Make predictions and evaluate
    y_pred = nb_classifier.predict(X_test)
    y_prob = nb_classifier.predict_proba(X_test)
    report = classification_report(y_test, y_pred, output_dict=True)
    auc_score = roc_auc_score(y_test, y_prob, multi_class='ovr')

    return accuracy_score(y_test, y_pred), pd.DataFrame(report).transpose(), y_pred, y_prob
```

```
In [25]: def train_and_evaluate_knn(X_train, X_test, y_train, y_test, n_neighbors=5):
    knn_classifier = KNeighborsClassifier(n_neighbors=n_neighbors)
    knn_classifier.fit(X_train, y_train)
    y_pred = knn_classifier.predict(X_test)
    y_prob = knn_classifier.predict_proba(X_test)

    report_df = pd.DataFrame(classification_report(y_test, y_pred, output_dict=True)).transpose()
    auc_score = roc_auc_score(y_test, y_prob, multi_class='ovr')

    return accuracy_score(y_test, y_pred), report_df, y_pred, y_prob
```

```
In [27]: def plot_roc_pr_curves(y_test, y_prob, title):
    fpr = []
    tpr = []
    roc_auc = []
    precision = []
    recall = []
    pr_auc = []

    for i in range(3):
        fpr[i], tpr[i] = roc_curve(y_test, y_prob[:, i], pos_label=i)
        roc_auc.append(roc_auc_fpr[i], tpr[i])
        precision[i], recall[i] = precision_recall_curve(y_test, y_prob[:, i], pos_label=i)
        pr_auc[i] = auc(recall[i], precision[i])

    plt.figure(figsize=(14, 6))

    plt.subplot(1, 2, 1)
    for i in range(3):
        plt.plot(fpr[i], tpr[i], label=f'Class {i} (area = {roc_auc[i]:.2f})')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve for {title}')
    plt.legend(loc='lower right')

    plt.subplot(1, 2, 2)
    for i in range(3):
        plt.plot(recall[i], precision[i], label=f'Class {i} (area = {pr_auc[i]:.2f})')
    plt.plot([0.0, 1.0], [0.0, 1.05], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title(f'Precision-Recall Curve for {title}')
    plt.legend(loc='lower left')

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

```
In [35]: def plot_confusion_matrix(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(10, 7))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No Diabetes', 'Prediabetes', 'Diabetes'], yticklabels=['No Diabetes', 'Prediabetes', 'Diabetes'])
    plt.xlabel('Predicted')
```

```
plt.ylabel('Actual')
plt.title(title)
plt.show()
```

```
In [36]: def display_results(accuracy, report, y_true, y_pred, title):
    print(title)
    print("Accuracy: {accuracy}")
    print("\n")
    print(report)
    print("\n")
    plot_confusion_matrix(y_true, y_pred, title)

In [37]: # Function to plot ROC curve using Yellowbrick
def plot_ROC_curve(model, xtrain, ytrain, xtest, ytest):
    visualizer = ROCAUC(model, encoder={0: 'Non-Diabetic', 1: 'Pre-Diabetic', 2: 'Diabetic'})
    visualizer.fit(xtrain, ytrain)
    visualizer.score(xtest, ytest)
    visualizer.show()
    return visualizer
```

Random Forest

Hyperparameter Tuning for RandomForest

```

[30]: # Define a reduced parameter grid for GridSearchCV
param_grid_rf_reduced = [
    'n_estimators': [100, 150],
    'max_depth': [10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
    'bootstrap': [True]
]

# Initialize the RandomForestClassifier
rf_clf_reduced = RandomForestClassifier(random_state=42)

# Perform GridSearchCV to find the best parameters for RandomForest using SMOTE balanced data
grid_search_rf_reduced = GridSearchCV(estimator=rf_clf_reduced, param_grid=param_grid_rf_reduced,
                                       cv=3, n_jobs=-1, verbose=2)

# Fit the grid search to the SMOTE balanced data
grid_search_rf_reduced.fit(X_oversampled, y_oversampled)

# Get the best parameters and the best model
best_params_rf_reduced_smote = grid_search_rf_reduced.best_params_
best_rf_clf_reduced_smote = grid_search_rf_reduced.best_estimator_
print("Best parameters for RandomForest (SMOTE) found: ", best_params_rf_reduced_smote)

# Perform GridSearchCV to find the best parameters for RandomForest using SMOTENN balanced data
grid_search_rf_reduced.fit(X_oversampled_enn, y_oversampled_enn)

# Get the best parameters and the best model
best_params_rf_reduced_enn = grid_search_rf_reduced.best_params_
best_rf_clf_reduced_enn = grid_search_rf_reduced.best_estimator_
print("Best parameters for RandomForest (SMOTENN) found: ", best_params_rf_reduced_enn)

# Fit the grid search to the undersampled data
grid_search_rf_reduced.fit(X_undersampled, y_undersampled)

# Get the best parameters and the best model
best_params_rf_reduced_undersampled = grid_search_rf_reduced.best_params_
best_rf_clf_reduced_undersampled = grid_search_rf_reduced.best_estimator_
print("Best parameters for RandomForest (undersampled) found: ", best_params_rf_reduced_undersampled)

Fitting 3 folds for each of 16 candidates, totaling 48 fits
Best parameters for RandomForest (SMOTE) found: {'bootstrap': True, 'max_depth': 20, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 150}
Fitting 3 folds for each of 16 candidates, totaling 48 fits
Best parameters for RandomForest (SMOTENN) found: {'bootstrap': True, 'max_depth': 20, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 150}
Fitting 3 folds for each of 16 candidates, totaling 48 fits
Best parameters for RandomForest (undersampled) found: {'bootstrap': True, 'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}

```

Best Parameters:

SMOTE and SMOTENN:

- bootstrap : True
 - max_depth : 20
 - min_samples_leaf : 1
 - min_samples_split : 2
 - n_estimators : 150

Undersampled:

- bootstrap : True
 - max_depth : 10
 - min_samples_leaf : 1
 - min_samples_split : 2

Analysis:

- 1. Bootstrap:**
 - Ensures sampling with replacement, promoting diverse trees and reducing overfitting.
 - 2. Max Depth:**
 - Depth of 20 for SMOTE/SMOTENN captures complex patterns; 10 for undersampled data to reduce overfitting.
 - 3. Min Samples Leaf and Split:**
 - Both set to defaults (1 and 2) to allow detailed tree growth and ensure nodes have enough samples before splitting.
 - 4. Number of Estimators:**
 - 150 trees for SMOTE/SMOTENN to reduce variance; 100 for undersampled to balance performance and computational efficiency.

Performance Considerations:

- **Consistency:** Identical parameters for SMOTE and SMOTENNN show robustness.
 - **Computational Load:** 150 trees and depth of 20 require adequate resources.
 - **Model Complexity:** Monitoring for overfitting is crucial, especially with undersampled data using fewer trees and depth

Evaluate the RandomForest Model

```
[39] from sklearn.metrics import accuracy_score, classification_report, roc_auc_score, roc_curve, au

# Predict on the test set using the tuned RandomForest model (SMOTE)
y_pred_best_rf_smote = best_rf_clf_reduced_smote.predict(X_test)
y_prob_best_rf_smote = best_rf_clf_reduced_smote.predict_proba(X_test)

# Predict on the test set using the tuned RandomForest model (SMOTENNN)
y_pred_best_rf_enn = best_rf_clf_reduced_enn.predict(X_test)
y_prob_best_rf_enn = best_rf_clf_reduced_enn.predict_proba(X_test)

# Predict on the test set using the tuned RandomForest model (undersampled)
y_pred_best_rf_undersampled = best_rf_clf_reduced_undersampled.predict(X_test)
```

```

# Evaluate the RandomForest model with SMOTE
accuracy_smote = accuracy_score(y_test, y_pred_best_rf_smote)
report_smote = classification_report(y_test, y_pred_best_rf_smote, output_dict=True)
display_results(accuracy_smote, pd.DataFrame(report_smote).transpose(), y_test, y_pred_best_rf_smote, "RandomForest (with SMOTE and Tuning)")
plot_roc_pr_curves(y_test, y_prob_best_rf_smote, "RandomForest (with SMOTE and Tuning)")

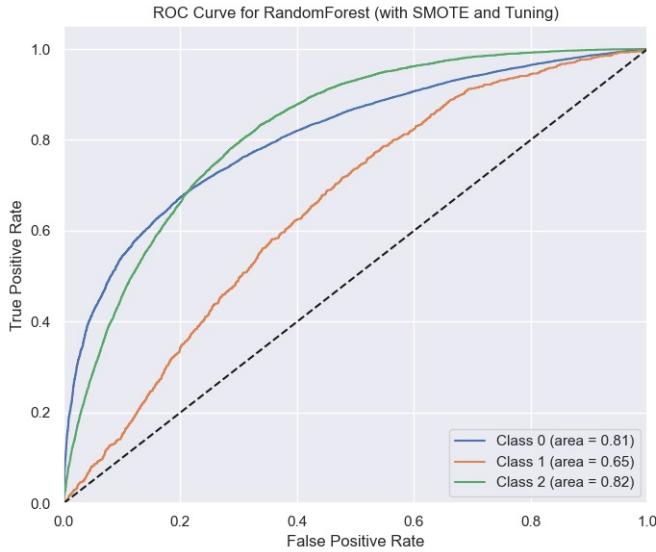
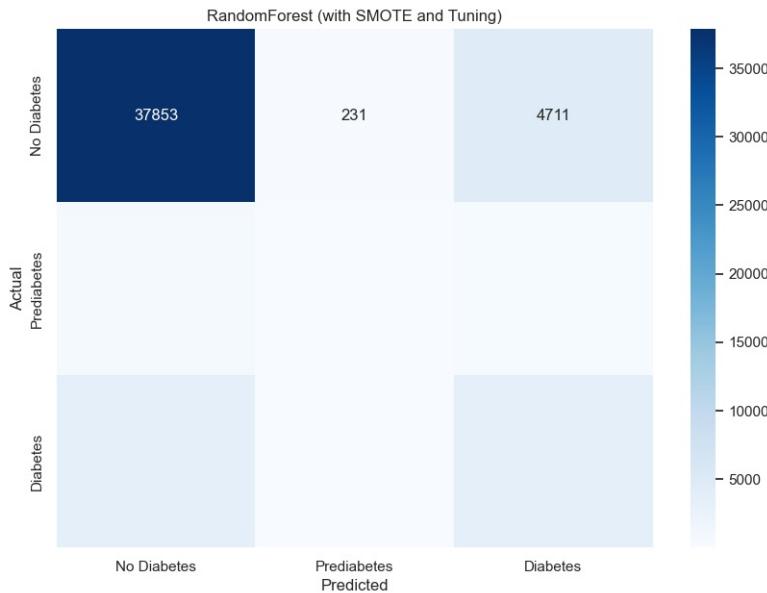
# Evaluate the RandomForest model with SMOTEEENN
accuracy_enn = accuracy_score(y_test, y_pred_best_rf_enn)
report_enn = classification_report(y_test, y_pred_best_rf_enn, output_dict=True)
display_results(accuracy_enn, pd.DataFrame(report_enn).transpose(), y_test, y_pred_best_rf_enn, "RandomForest (with SMOTEEENN and Tuning)")
plot_roc_pr_curves(y_test, y_prob_best_rf_enn, "RandomForest (with SMOTEEENN and Tuning)")

# Evaluate the RandomForest model with undersampling
accuracy_undersampled = accuracy_score(y_test, y_pred_best_rf_undersampled)
report_undersampled = classification_report(y_test, y_pred_best_rf_undersampled, output_dict=True)
display_results(accuracy_undersampled, pd.DataFrame(report_undersampled).transpose(), y_test, y_pred_best_rf_undersampled, "RandomForest (with undersampling and Tuning)")
plot_roc_pr_curves(y_test, y_prob_best_rf_undersampled, "RandomForest (with undersampling and Tuning)")

RandomForest (with SMOTE and Tuning)
Accuracy: 0.813643172500784

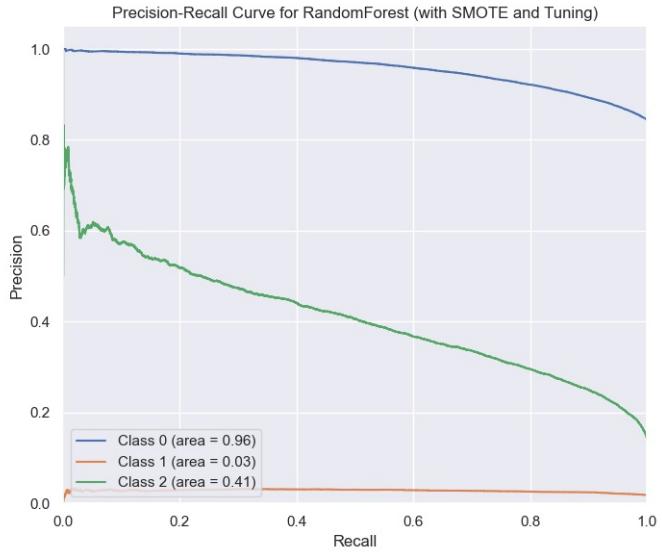
```

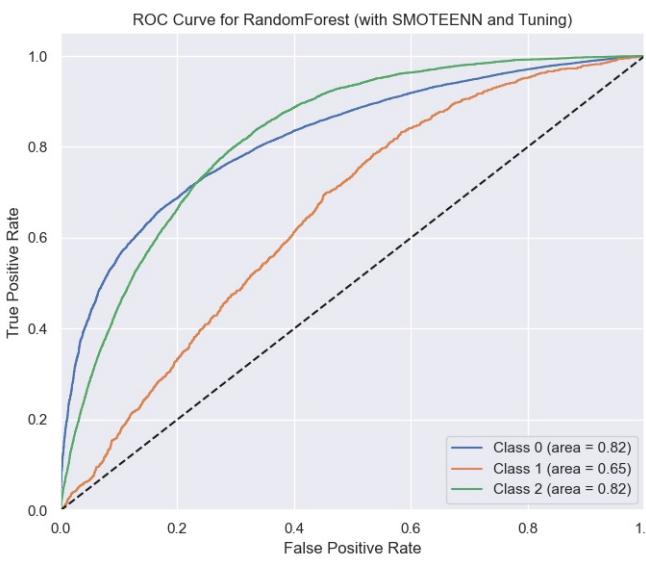
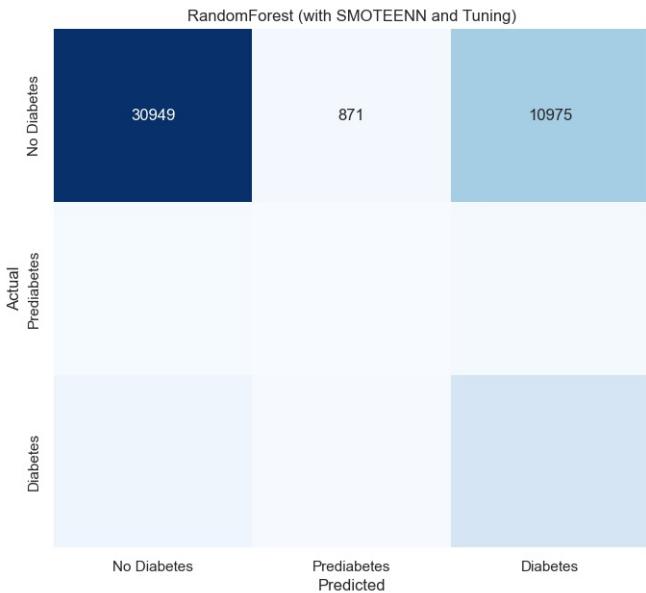
	precision	recall	f1-score	support
0.0	0.901369	0.884519	0.892865	42795.000000
1.0	0.029605	0.009534	0.014423	944.000000
2.0	0.405239	0.488638	0.443048	6997.000000
accuracy	0.813643	0.813643	0.813643	0.813643
macro avg	0.445404	0.460897	0.450112	50736.000000
weighted avg	0.816728	0.813643	0.814486	50736.000000



RandomForest (with SMOTEEENN and Tuning)
Accuracy: 0.7153894670450962

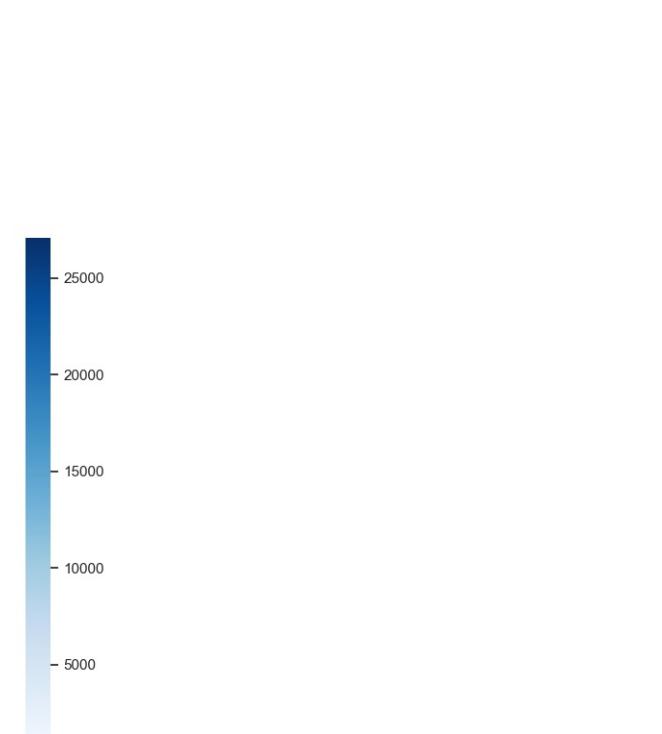
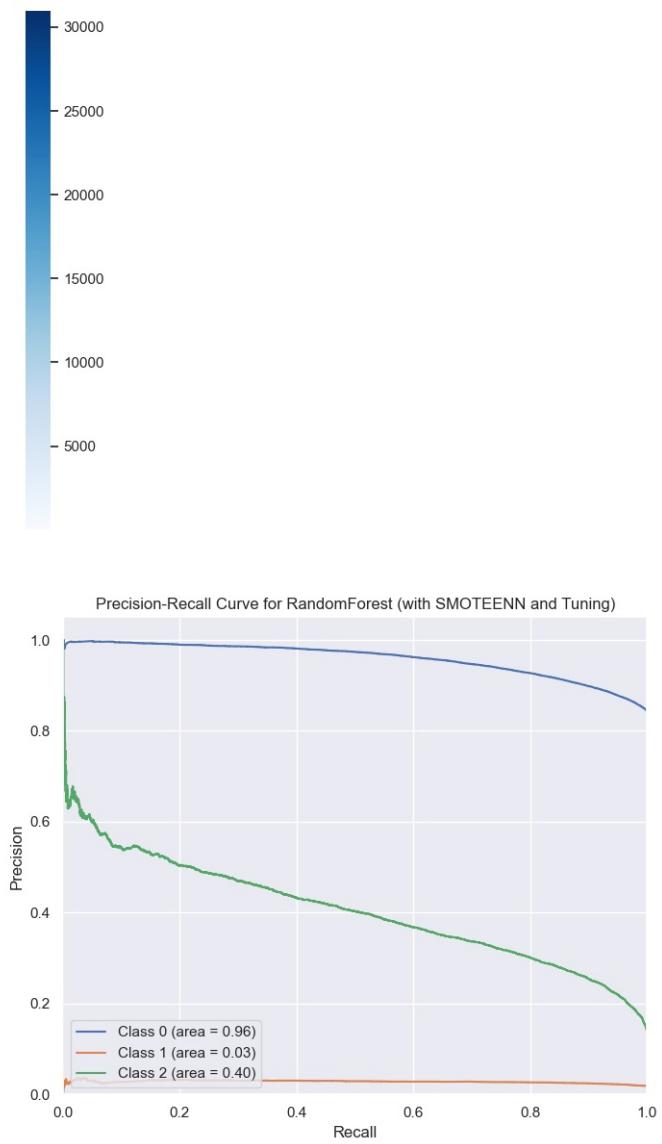
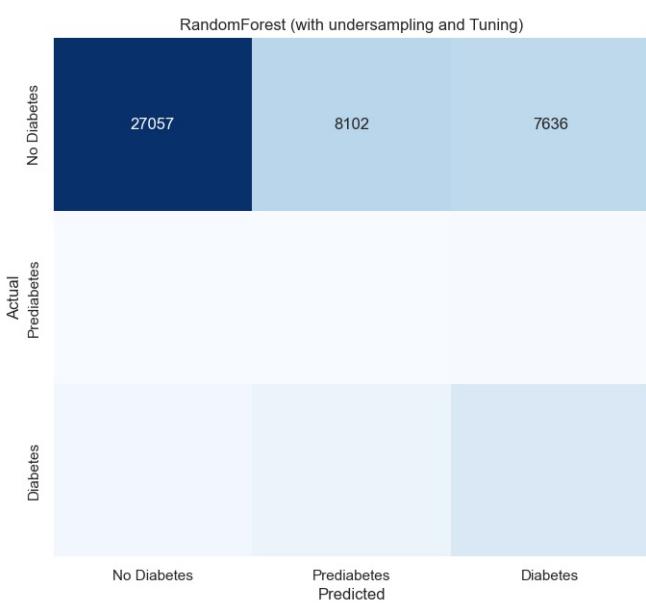
	precision	recall	f1-score	support
0.0	0.943768	0.723192	0.818867	42795.000000
1.0	0.027548	0.031780	0.029513	944.000000
2.0	0.315474	0.759897	0.445851	6997.000000
accuracy	0.715389	0.715389	0.715389	0.715389
macro avg	0.428930	0.504956	0.431417	50736.000000
weighted avg	0.840073	0.715389	0.752754	50736.000000

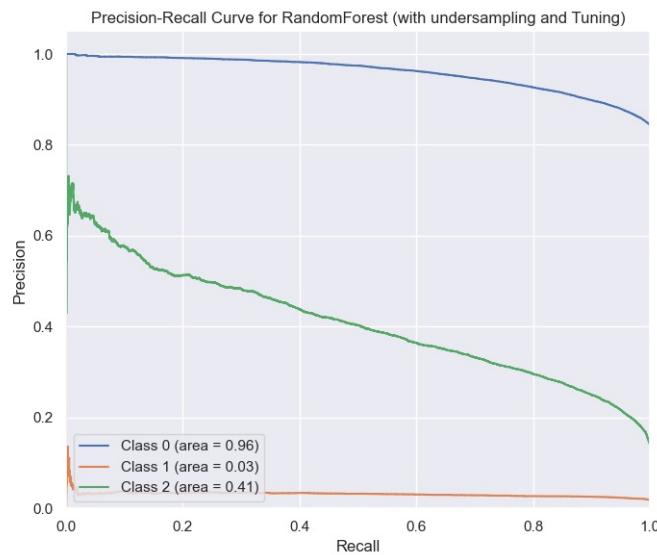
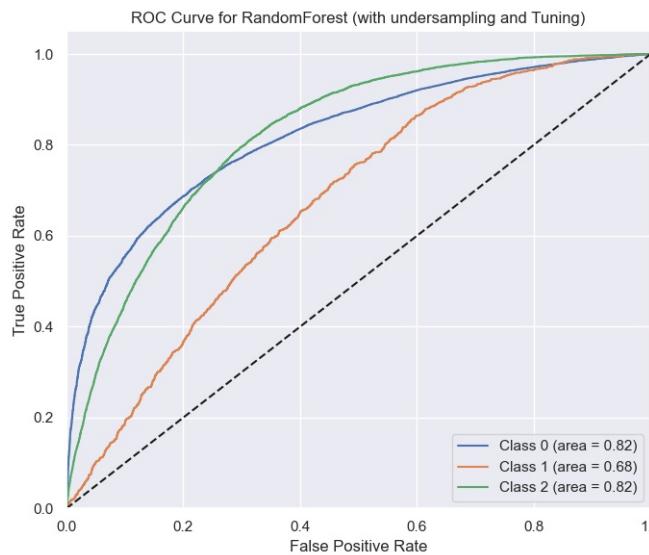




RandomForest (with undersampling and Tuning)
Accuracy: 0.6226545253863135

```
precision    recall   f1-score   support
0.0      0.956382  0.632247  0.761247  42795.000000
1.0      0.031586  0.342161  0.057833  944.000000
2.0      0.344627  0.601829  0.438281  6997.000000
accuracy          0.622655  0.622655  0.622655
macro avg      0.444198  0.525412  0.419120  50736.000000
weighted avg     0.854808  0.622655  0.703619  50736.000000
```





Model Evaluation with SMOTE

RandomForest Accuracy (with SMOTE and Tuning): 81.36%

1. **High Precision for "No Diabetes" (0):**
 - The model exhibits high precision (0.90), suggesting a minimal false positive rate in predicting the absence of diabetes.
2. **Low Performance for "Prediabetes" (1):**
 - Both precision and recall scores are underwhelming for this category, indicating challenges in accurately pinpointing prediabetes instances, possibly due to a limited sample size.
3. **Moderate Performance for "Diabetes" (2):**
 - The model demonstrates moderate precision (0.40) and recall (0.49) for diabetes cases, leaving room for enhancement.

Confusion Matrix Analysis:

- The majority of predictions align correctly with the "No Diabetes" class.
- Notable misclassifications between instances labeled as "No Diabetes" and those identified as "Diabetes."

ROC and Precision-Recall Analysis for SMOTE

ROC Curve:

- ROC-AUC scores are 0.81 for class 0, 0.65 for class 1, and 0.82 for class 2.

Precision-Recall Curve:

- Precision-Recall curves show high precision for class 0 but lower for classes 1 and 2.

Model Evaluation with SMOTEEENN

RandomForest Accuracy (with SMOTEEENN and Tuning): 71.54%

1. **Improved Recall for "Diabetes" (2):**
 - The recall rate for diabetes cases is improved at 0.76 compared to the SMOTE model, indicating better identification of true instances of diabetes.
2. **Lower Performance for "No Diabetes" (0):**
 - Precision and recall for this class are lower than the SMOTE model, suggesting more false positives and false negatives.
3. **Consistent Low Performance for "Prediabetes" (1):**
 - Similar to the SMOTE model, prediabetes remains difficult to predict accurately.

Confusion Matrix Analysis:

- More instances of incorrectly classifying cases of no diabetes as diabetes or prediabetes (false negatives).
- Increase in correctly identifying cases of diabetes (true positives).

ROC and Precision-Recall Analysis for SMOTEEENN

ROC Curve:

- ROC-AUC scores are 0.82 for class 0, 0.65 for class 1, and 0.82 for class 2.

Precision-Recall Curve:

- Precision-Recall curves show high precision for class 0 but lower for classes 1 and 2.

Model Evaluation with Undersampling

RandomForest Accuracy (with undersampling and Tuning): 62.27%

1. **High Precision for "No Diabetes" (0):**
 - The model shows high precision (0.96) but low recall (0.63), indicating it often predicts non-diabetic correctly but misses many true non-diabetic cases.
2. **Low Precision for "Prediabetes" (1):**
 - Precision is very low at 0.03, with recall at 0.34.
3. **Moderate Precision for "Diabetes" (2):**
 - Precision is 0.34, and recall is 0.60, indicating the model can identify diabetes cases but with a moderate false positive rate.

Confusion Matrix Analysis:

- High misclassifications between classes, particularly for prediabetes.

ROC and Precision-Recall Analysis for Undersampling

ROC Curve:

- ROC-AUC scores are 0.82 for class 0, 0.68 for class 1, and 0.82 for class 2.

Precision-Recall Curve:

- Precision-Recall curves show high precision for class 0 but lower for classes 1 and 2.

General Insights

1. **Accuracy vs. Recall Trade-off:**
 - The SMOTE model has higher overall accuracy but lower recall for diabetes.
 - The SMOTEEENN model has better recall for diabetes, making it more suitable for scenarios where identifying all potential diabetes cases is crucial, even if it means lower overall accuracy.

2. Class Imbalance Challenge:

- Both models struggle with the "Prediabetes" class due to its small sample size. Further data collection or different modeling approaches may be needed to improve predictions for this class.

3. Business Implications:

- **SMOTE Model:** Higher accuracy can lead to better general performance in identifying non-diabetic individuals, reducing unnecessary interventions.
- **SMOTEENN Model:** Improved recall for diabetes suggests better early detection, potentially leading to timely intervention and better patient outcomes despite a higher rate of false positives.

4. Model Selection:

- The choice between SMOTE and SMOTEENN depends on the specific needs of the healthcare setting. If the priority is to minimize missed diabetes cases, SMOTEENN might be preferred. For a balanced approach with higher overall

References:

1. Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence Research, 16, 321-357. <https://doi.org/10.1613/jair.953>
2. He, H., & Garcia, E. A. (2009). Learning from Imbalanced Data. IEEE Transactions on Knowledge and Data Engineering, 21(9), 1263-1284. <https://doi.org/10.1109/TKDE.2008.239>
3. Breiman, L. (2001). Random Forests. Machine Learning, 45(1), 5-32. <https://doi.org/10.1023/A:1010933404324>
4. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830. Retrieved from <http://www.jmlr.org>

SVM

```
# Slicing the data - 10,000 rows for SVM
num_rows = 10_000

X_train_slice = X_train.sample(n=num_rows, random_state=42)
y_train_slice = y_train.loc[X_train_slice.index]

# Oversampled data
X_oversampled_slice = X_oversampled.sample(n=num_rows, random_state=42)
y_oversampled_slice = y_oversampled.loc[X_oversampled_slice.index]

# Undersampled data
X_undersampled_slice = X_undersampled.sample(n=num_rows, random_state=42)
y_undersampled_slice = y_undersampled.loc[X_undersampled_slice.index]

# SMOTEENN data
X_oversampled_enn_slice = X_oversampled_enn.sample(n=num_rows, random_state=42)
y_oversampled_enn_slice = y_oversampled_enn.loc[X_oversampled_enn_slice.index]

# Train and evaluate SVM models
# SVM + Original Data
svm_model = SVC(probability=True, random_state=42)
svm_model.fit(X_train_slice, y_train_slice)
y_pred_original = svm_model.predict(X_test)
y_prob_original = svm_model.predict_proba(X_test)
accuracy_original = accuracy_score(y_test, y_pred_original)
report_original = classification_report(y_test, y_pred_original, output_dict=True)
display_results(accuracy_original, pd.DataFrame(report_original).transpose(), y_test, y_pred_original, "SVM (Original)")
plot_roc_pr_curves(y_test, y_prob_original, "SVM (Original)")

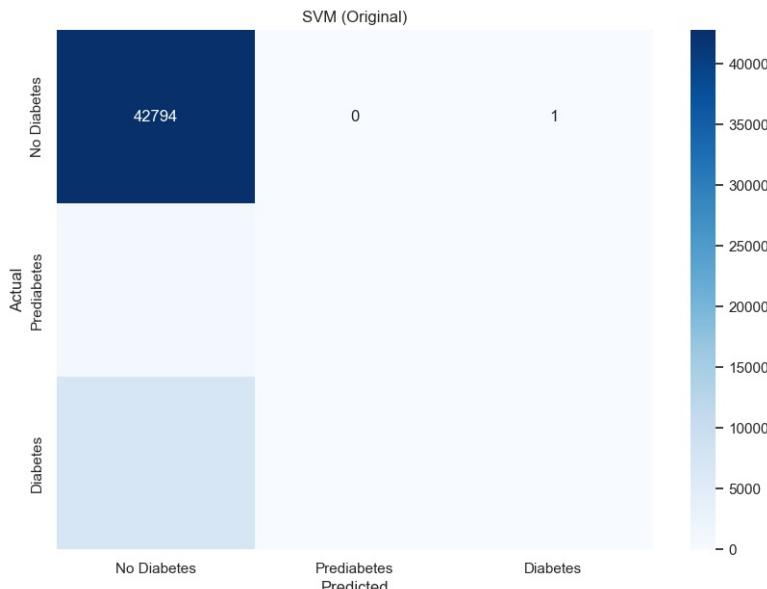
# SVM + Oversampled Data
svm_model.fit(X_oversampled_slice, y_oversampled_slice)
y_pred_oversampled = svm_model.predict(X_test)
y_prob_oversampled = svm_model.predict_proba(X_test)
accuracy_oversampled = accuracy_score(y_test, y_pred_oversampled)
report_oversampled = classification_report(y_test, y_pred_oversampled, output_dict=True)
display_results(accuracy_oversampled, pd.DataFrame(report_oversampled).transpose(), y_test, y_pred_oversampled, "SVM (Oversampled)")
plot_roc_pr_curves(y_test, y_prob_oversampled, "SVM (Oversampled)")

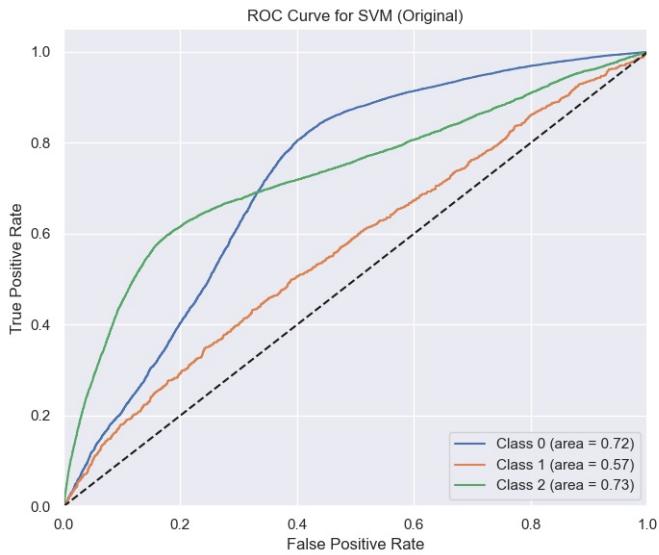
# SVM + Undersampled Data
svm_model.fit(X_undersampled_slice, y_undersampled_slice)
y_pred_undersampled = svm_model.predict(X_test)
y_prob_undersampled = svm_model.predict_proba(X_test)
accuracy_undersampled = accuracy_score(y_test, y_pred_undersampled)
report_undersampled = classification_report(y_test, y_pred_undersampled, output_dict=True)
display_results(accuracy_undersampled, pd.DataFrame(report_undersampled).transpose(), y_test, y_pred_undersampled, "SVM (Undersampled)")
plot_roc_pr_curves(y_test, y_prob_undersampled, "SVM (Undersampled)")

# SVM + SMOTEENN Data
svm_model.fit(X_oversampled_enn_slice, y_oversampled_enn_slice)
y_pred_smoteenn = svm_model.predict(X_test)
y_prob_smoteenn = svm_model.predict_proba(X_test)
accuracy_smoteenn = accuracy_score(y_test, y_pred_smoteenn)
report_smoteenn = classification_report(y_test, y_pred_smoteenn, output_dict=True)
display_results(accuracy_smoteenn, pd.DataFrame(report_smoteenn).transpose(), y_test, y_pred_smoteenn, "SVM (SMOTEENN)")
plot_roc_pr_curves(y_test, y_prob_smoteenn, "SVM (SMOTEENN)")

SVM (Original)
Accuracy: 0.8435036266162094
```

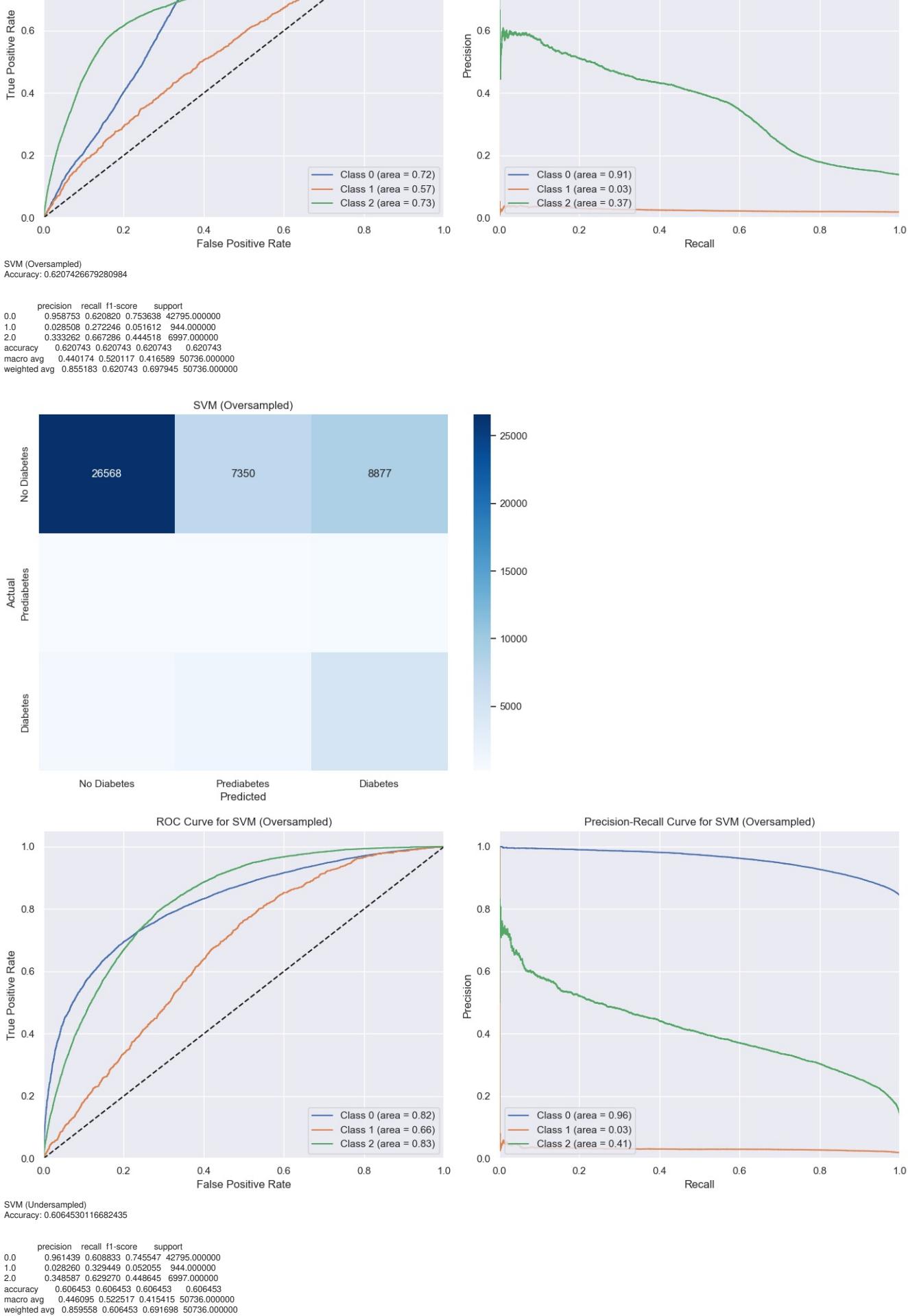
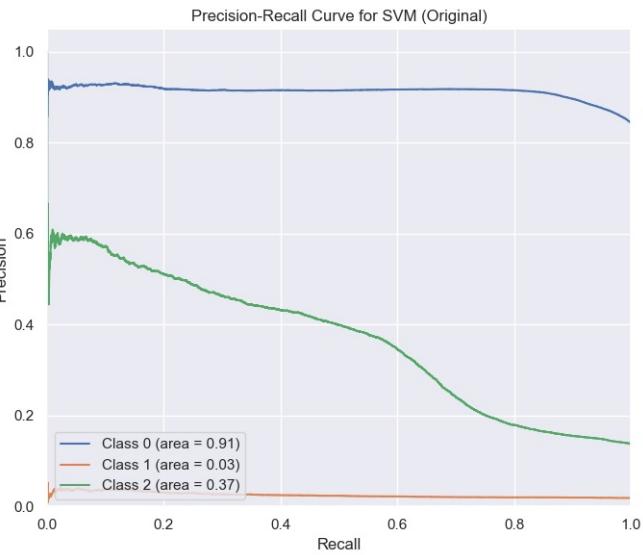
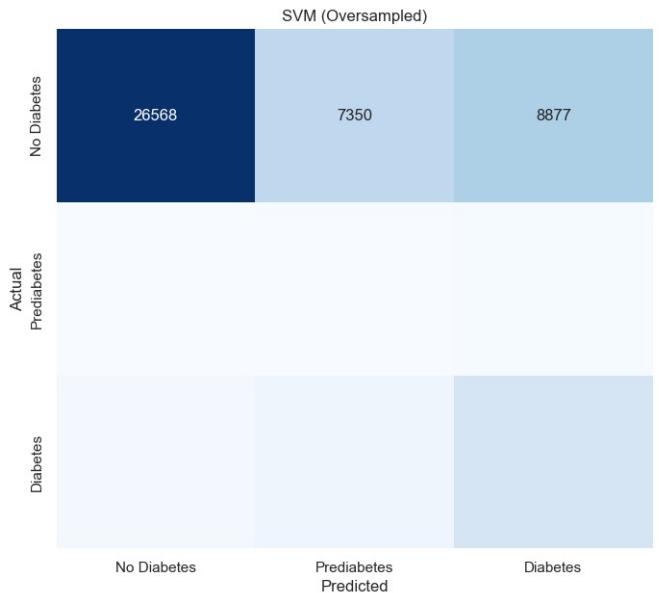
	precision	recall	f1-score	support
0.0	0.843514	0.999977	0.915106	42795.000000
1.0	0.000000	0.000000	0.000000	944.000000
2.0	0.666667	0.000286	0.000571	6997.000000
accuracy	0.843504	0.843504	0.843504	0.843504
macro avg	0.503394	0.333421	0.305226	50736.000000
weighted avg	0.803431	0.843504	0.771956	50736.000000



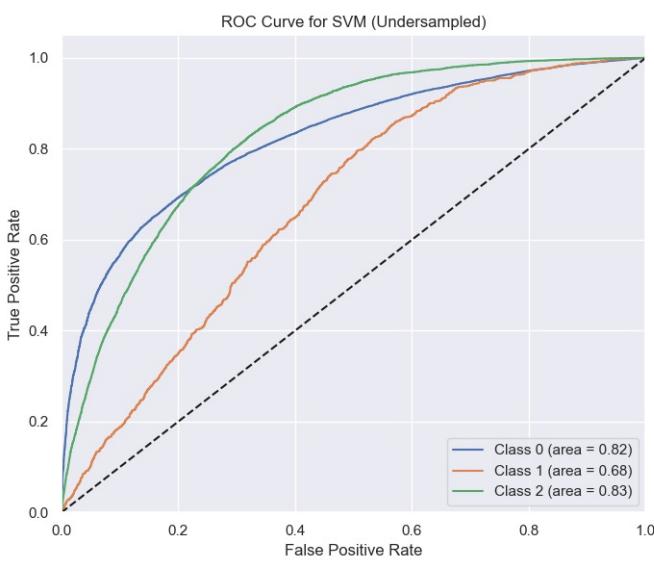
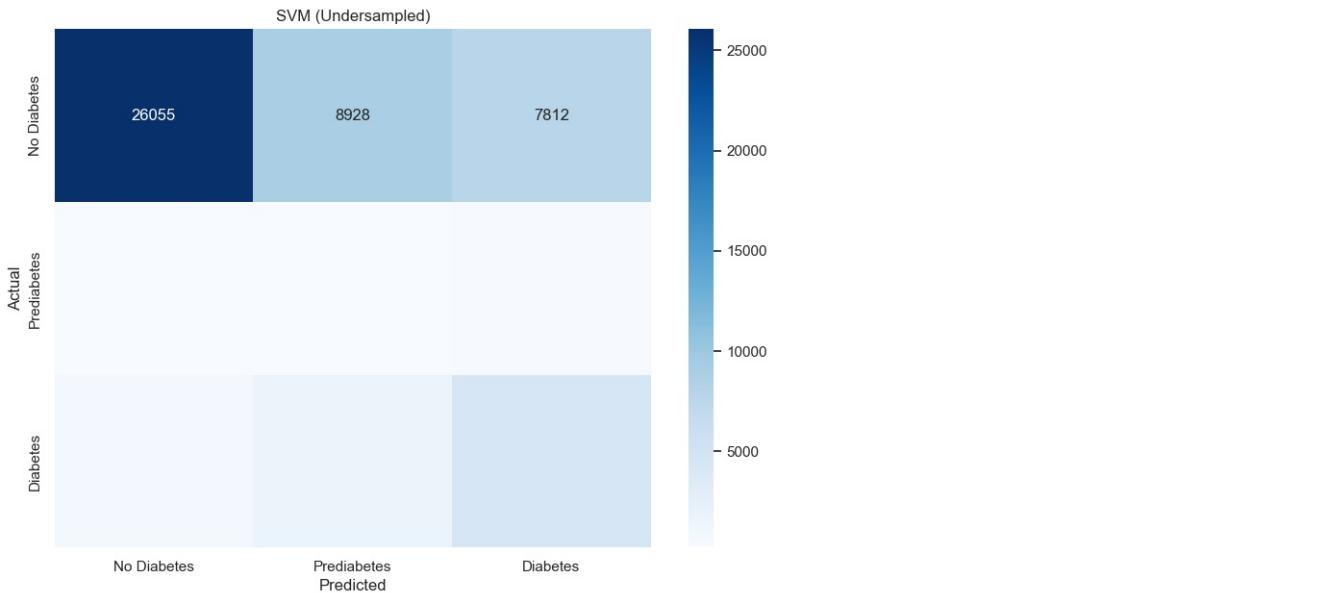


SVM (Oversampled)
Accuracy: 0.6207426679280984

```
precision    recall  f1-score   support
0.0      0.958753  0.620920  0.753638  42795.000000
1.0      0.028508  0.272246  0.051612  944.000000
2.0      0.333262  0.667286  0.444518  6997.000000
accuracy   0.620743  0.620743  0.620743  0.620743
macro avg   0.440174  0.520117  0.416589  50736.000000
weighted avg 0.855183  0.620743  0.697945  50736.000000
```

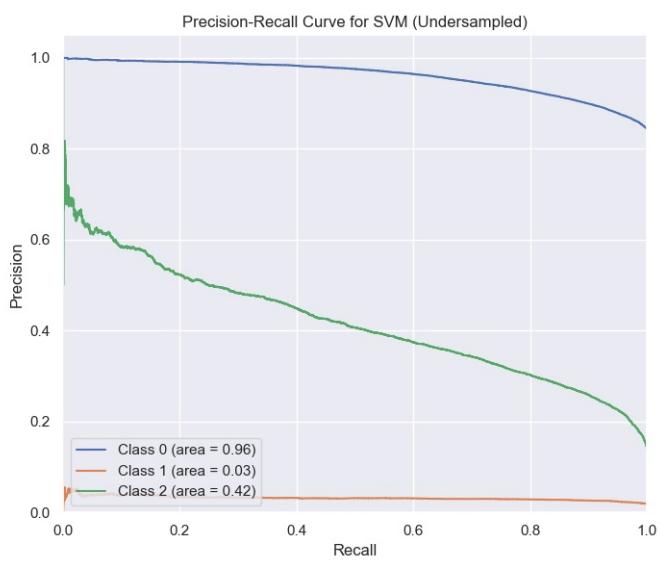
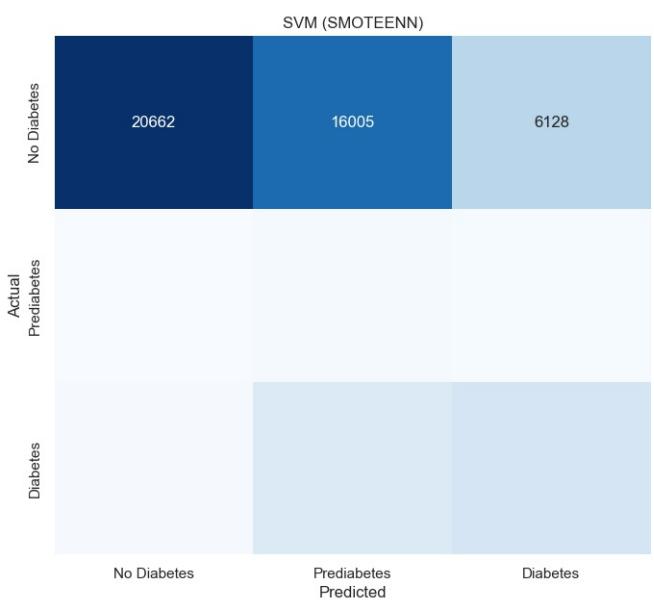


```
precision    recall  f1-score   support
0.0      0.961439  0.609833  0.745547  42795.000000
1.0      0.028620  0.329449  0.052055  944.000000
2.0      0.348587  0.629270  0.448645  6997.000000
accuracy   0.606453  0.606453  0.606453  0.606453
macro avg   0.446095  0.522517  0.415415  50736.000000
weighted avg 0.859558  0.606453  0.691698  50736.000000
```

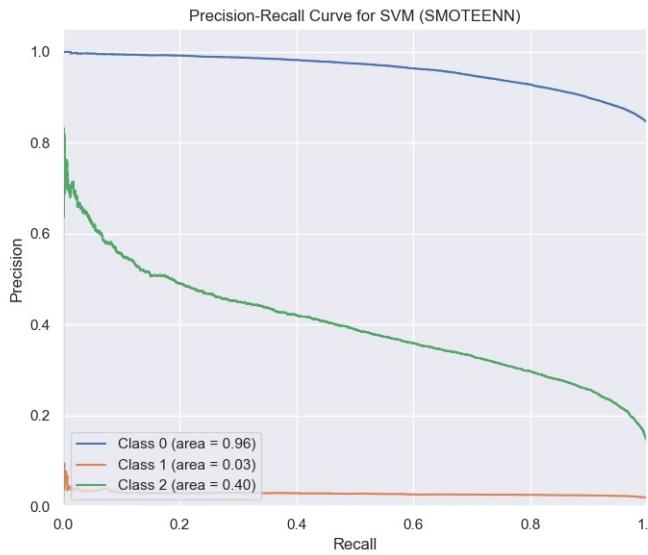
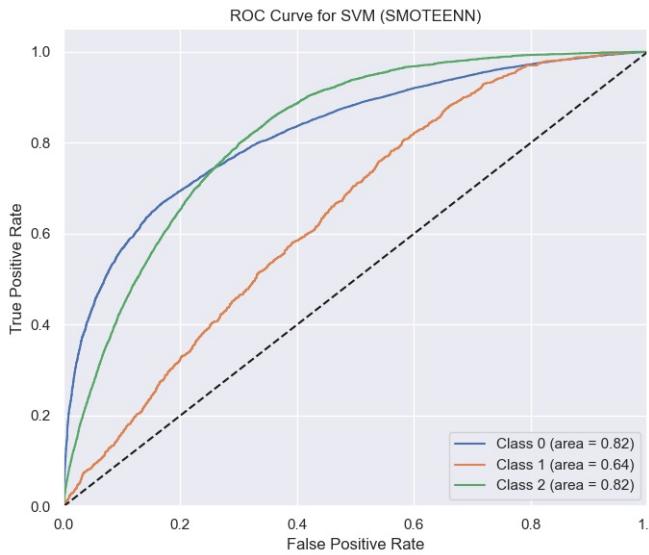


SVM (SMOTEENN)
Accuracy: 0.4899085461999369

```
precision    recall   f1-score   support
0.0      0.975129  0.482813  0.645849  42795.000000
1.0      0.024985  0.512712  0.047647  944.000000
2.0      0.364619  0.530227  0.432099  6997.000000
accuracy          0.489909  0.489909  0.489909
macro avg      0.454911  0.508584  0.375198  50736.000000
weighted avg     0.873255  0.489909  0.605240  50736.000000
```



```
precision    recall   f1-score   support
0.0      0.975129  0.482813  0.645849  42795.000000
1.0      0.024985  0.512712  0.047647  944.000000
2.0      0.364619  0.530227  0.432099  6997.000000
accuracy          0.489909  0.489909  0.489909
macro avg      0.454911  0.508584  0.375198  50736.000000
weighted avg     0.873255  0.489909  0.605240  50736.000000
```



Model Evaluation with SVM

SVM (Original Data)

- Accuracy: 84.35%

1. High Precision for "No Diabetes" (0): The model shows excellent precision (0.84), indicating a low false positive rate in predicting the absence of diabetes.
2. Zero Recall for "Prediabetes" (1) and "Diabetes" (2): The model fails to identify any instances of prediabetes or diabetes, as reflected by the zero recall for these classes.

Confusion Matrix Analysis:

- The majority of predictions correctly align with the "No Diabetes" class.
- There are almost no correct predictions for the "Prediabetes" and "Diabetes" classes, indicating severe class imbalance handling issues.

ROC and Precision-Recall Analysis for SVM (Original Data):

- ROC Curve: ROC-AUC scores are 0.72 for class 0, 0.57 for class 1, and 0.73 for class 2.
- Precision-Recall Curve: Precision is high for class 0 but significantly lower for classes 1 and 2.

SVM (Oversampled Data)

- Accuracy: 62.07%

1. High Precision for "No Diabetes" (0): The model maintains high precision (0.96), but lower recall (0.62) indicates many false negatives.
2. Improved Recall for "Diabetes" (2): The model shows improved recall (0.67) for diabetes cases, highlighting better identification of true instances.
3. Low Performance for "Prediabetes" (1): The recall for prediabetes is low (0.27), suggesting difficulties in accurately identifying these instances.

Confusion Matrix Analysis:

- The model shows more balanced predictions across classes compared to the original data model.
- There are still significant misclassifications, especially between "No Diabetes" and the other classes.

ROC and Precision-Recall Analysis for SVM (Oversampled Data):

- ROC Curve: ROC-AUC scores are 0.82 for class 0, 0.66 for class 1, and 0.83 for class 2.
- Precision-Recall Curve: Precision is high for class 0 but lower for classes 1 and 2.

SVM (Undersampled Data)

- Accuracy: 60.65%

1. High Precision for "No Diabetes" (0): The model maintains high precision (0.96) with lower recall (0.61), indicating many false negatives.
2. Improved Recall for "Diabetes" (2): The recall for diabetes is better at 0.63.
3. Low Performance for "Prediabetes" (1): Precision and recall for prediabetes remain low.

Confusion Matrix Analysis:

- High misclassifications, particularly between "No Diabetes" and "Prediabetes."

ROC and Precision-Recall Analysis for SVM (Undersampled Data):

- ROC Curve: ROC-AUC scores are 0.82 for class 0, 0.68 for class 1, and 0.83 for class 2.
- Precision-Recall Curve: Precision is high for class 0 but lower for classes 1 and 2.

SVM (SMOTEENN Data)

- Accuracy: 48.99%

1. High Precision for "No Diabetes" (0): The model shows high precision (0.97) but lower recall (0.48), indicating many false negatives.
2. Moderate Performance for "Diabetes" (2): The recall for diabetes is moderate at 0.53.
3. Improved Recall for "Prediabetes" (1): The model shows improved recall (0.51) for prediabetes cases, highlighting better identification of true instances.

Confusion Matrix Analysis:

- The model shows significant misclassifications across all classes.

ROC and Precision-Recall Analysis for SVM (SMOTEENN Data):

- ROC Curve: ROC-AUC scores are 0.82 for class 0, 0.64 for class 1, and 0.82 for class 2.
- Precision-Recall Curve: Precision is high for class 0 but lower for classes 1 and 2.

General Insights

1. Accuracy vs. Recall Trade-off:

- The SVM model with original data has the highest overall accuracy but fails to recall any instances of prediabetes or diabetes.
- Oversampling and undersampling improve recall for diabetes but at the cost of overall accuracy.

2. Class Imbalance Challenge:

- The models struggle with accurately predicting prediabetes due to its small sample size.
- Different modeling approaches or further data collection may be needed to improve predictions for this class.

3. Business Implications:

- Original Model: Higher accuracy can lead to better general performance in identifying non-diabetic individuals, reducing unnecessary interventions.

- **Oversampled/Undersampled Models:** Improved recall for diabetes suggests better early detection, potentially leading to timely intervention and better patient outcomes despite a higher rate of false positives.

4. Model Selection:

- The choice between original and resampled models depends on the specific needs of the healthcare setting. If the priority is to minimize missed diabetes cases, resampled models might be preferred. For a balanced approach with high more suitable.

Naive Bayes

```
In [53]: # Train and evaluate Naive Bayes models
# Naive Bayes + Original Data
accuracy, report, y_pred_original, y_prob_original = train_and_evaluate_nb(X_train, X_test, y_train, y_test)
display_results(accuracy, report, y_test, y_pred_original, "Naive Bayes (Original)")
plot_roc_pr_curves(y_test, y_prob_original, "Naive Bayes (Original)")

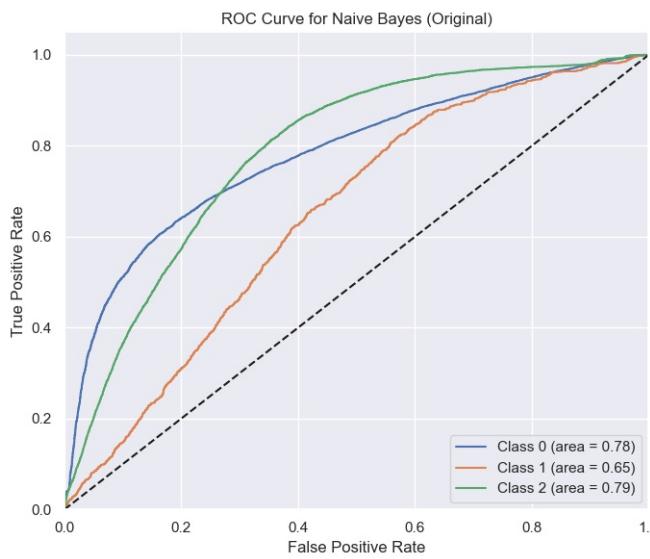
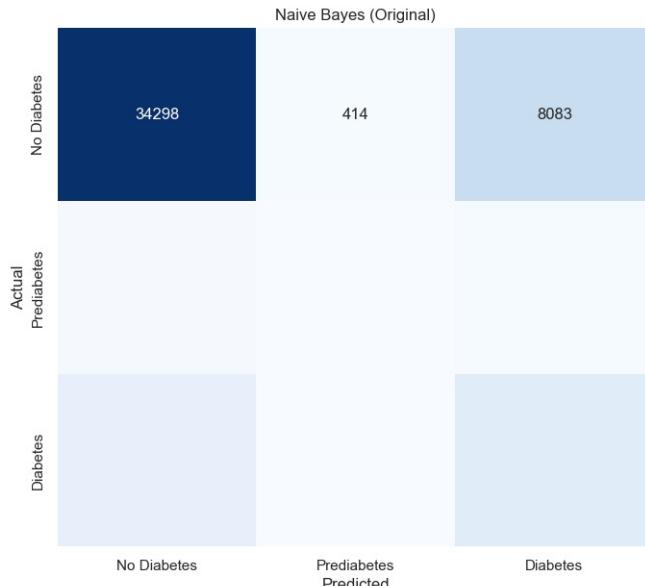
# Naive Bayes + Oversampled Data
accuracy, report, y_pred_oversampled, y_prob_oversampled = train_and_evaluate_nb(X_oversampled, X_test, y_oversampled, y_test)
display_results(accuracy, report, y_test, y_pred_oversampled, "Naive Bayes (Oversampled)")
plot_roc_pr_curves(y_test, y_prob_oversampled, "Naive Bayes (Oversampled)")

# Naive Bayes + Undersampled Data
accuracy, report, y_pred_undersampled, y_prob_undersampled = train_and_evaluate_nb(X_undersampled, X_test, y_undersampled, y_test)
display_results(accuracy, report, y_test, y_pred_undersampled, "Naive Bayes (Undersampled)")
plot_roc_pr_curves(y_test, y_prob_undersampled, "Naive Bayes (Undersampled)")

# Naive Bayes + SMOTEENN Data
accuracy, report, y_pred_smoteenn, y_prob_smoteenn = train_and_evaluate_nb(X_oversampled_enn, X_test, y_oversampled_enn, y_test)
display_results(accuracy, report, y_test, y_pred_smoteenn, "Naive Bayes (SMOTEENN)")
plot_roc_pr_curves(y_test, y_prob_smoteenn, "Naive Bayes (SMOTEENN)")
```

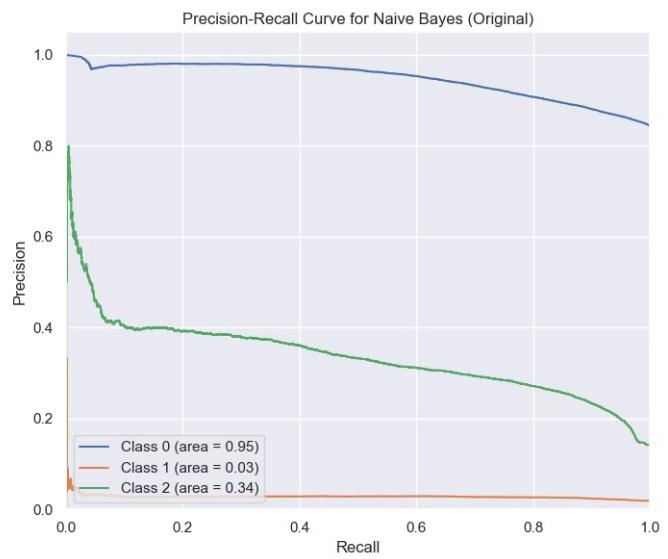
Naive Bayes (Original)
Accuracy: 0.7537251655629139

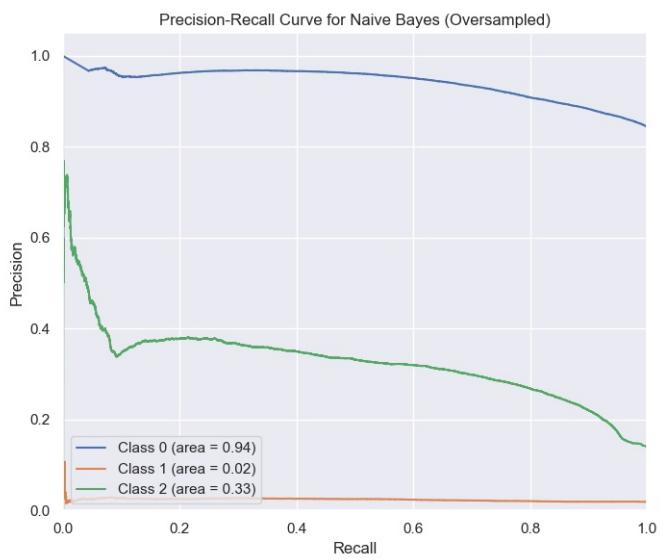
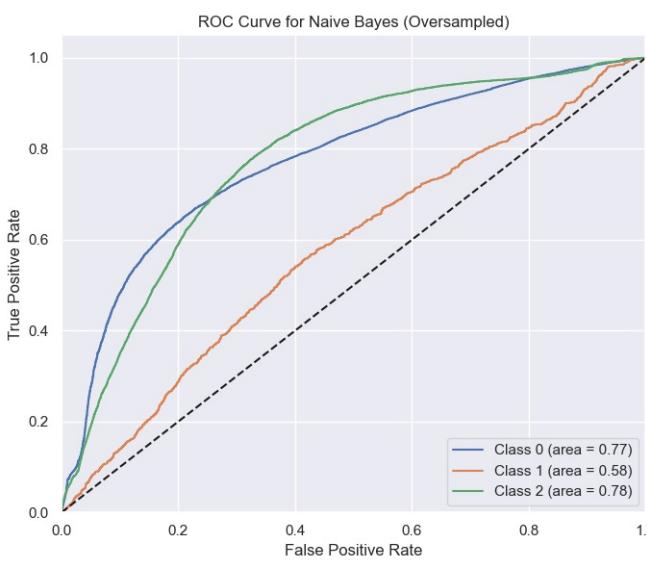
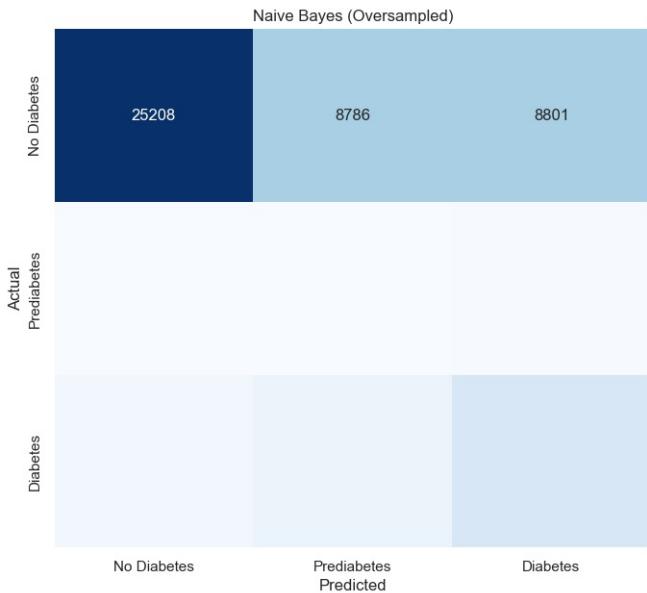
	precision	recall	f1-score	support
0.0	0.907354	0.801449	0.851120	42795.000000
1.0	0.038182	0.022246	0.028112	944.000000
2.0	0.316648	0.560526	0.404685	6997.000000
accuracy	0.753725	0.753725	0.753725	0.753725
macro avg	0.420728	0.461407	0.427972	50736.000000
weighted avg	0.809718	0.753725	0.774239	50736.000000



Naive Bayes (Oversampled)
Accuracy: 0.5860927152317881

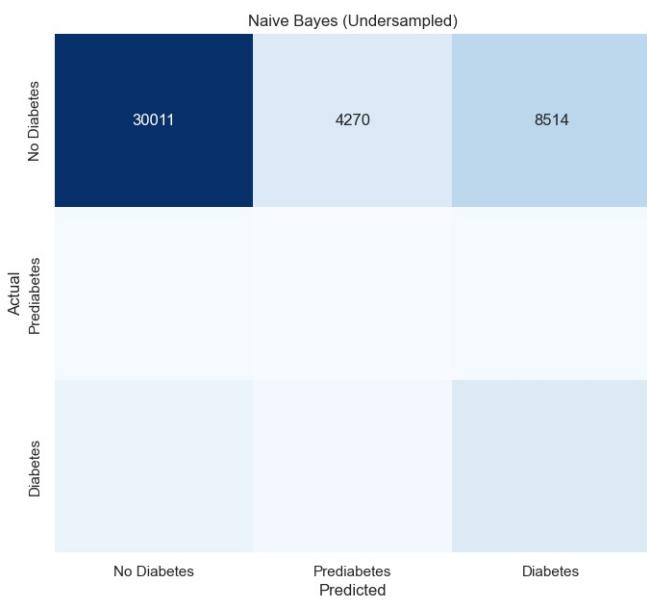
	precision	recall	f1-score	support
0.0	0.951604	0.589041	0.727661	42795.000000
1.0	0.025769	0.294492	0.047392	944.000000
2.0	0.315797	0.607403	0.415546	6997.000000
accuracy	0.586093	0.586093	0.586093	0.586093
macro avg	0.431057	0.496978	0.396866	50736.000000
weighted avg	0.846694	0.586093	0.671960	50736.000000

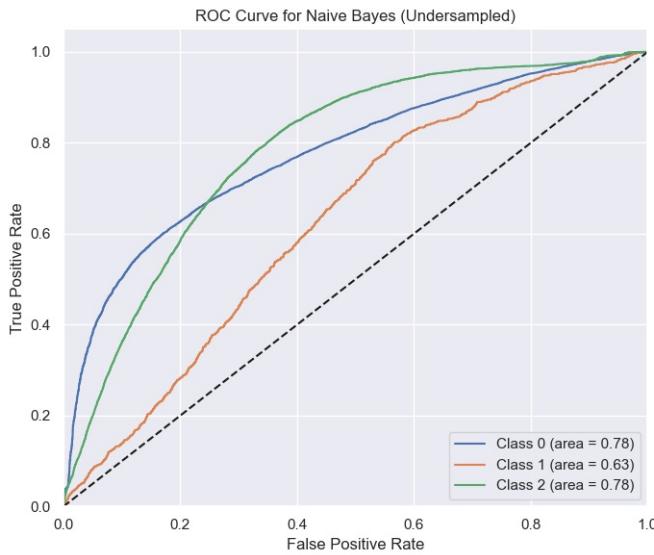




Naive Bayes (Undersampled)
Accuracy: 0.6767975402081362

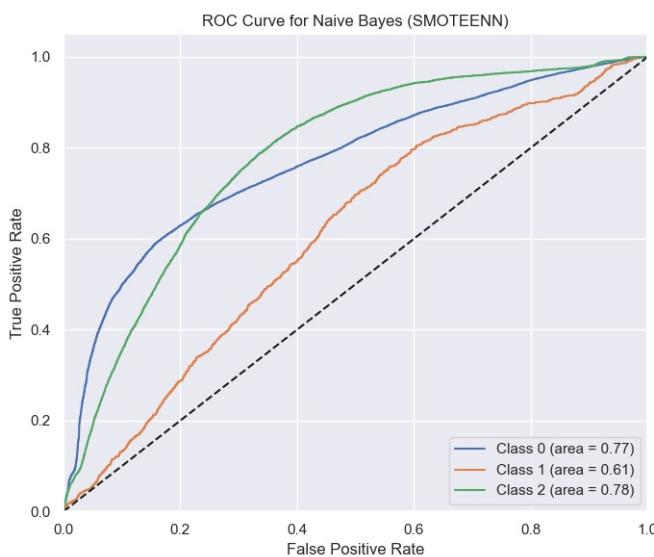
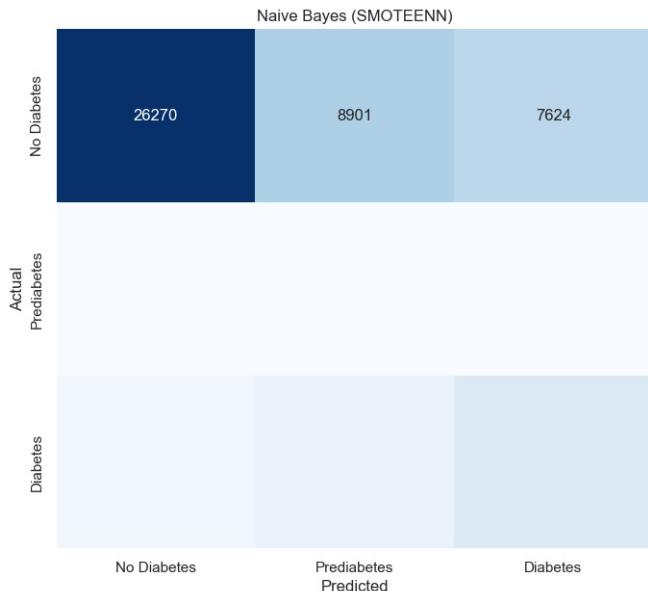
```
precision    recall   f1-score   support
0.0      0.925865  0.701274  0.798069  42795.000000
1.0      0.026623  0.147246  0.045093  944.000000
2.0      0.319670  0.598542  0.416758  6997.000000
accuracy          0.676798  0.676798  0.676798
macro avg      0.424053  0.482354  0.419974  50736.000000
weighted avg     0.825534  0.676798  0.731473  50736.000000
```





Naive Bayes (SMOTEENN)
Accuracy: 0.5991406496373384

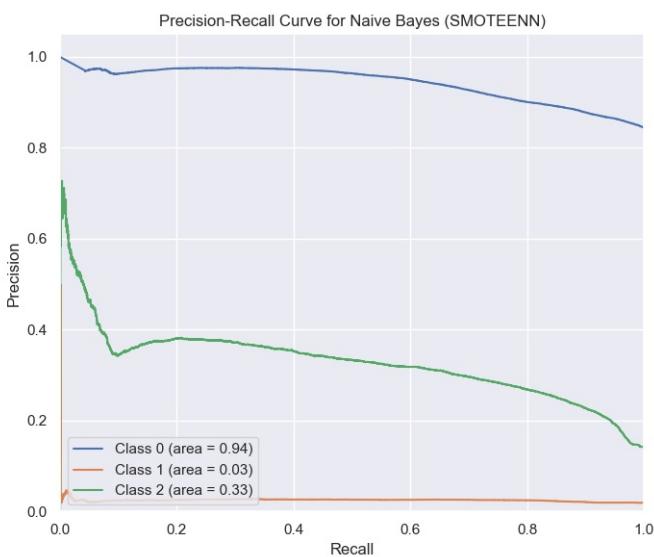
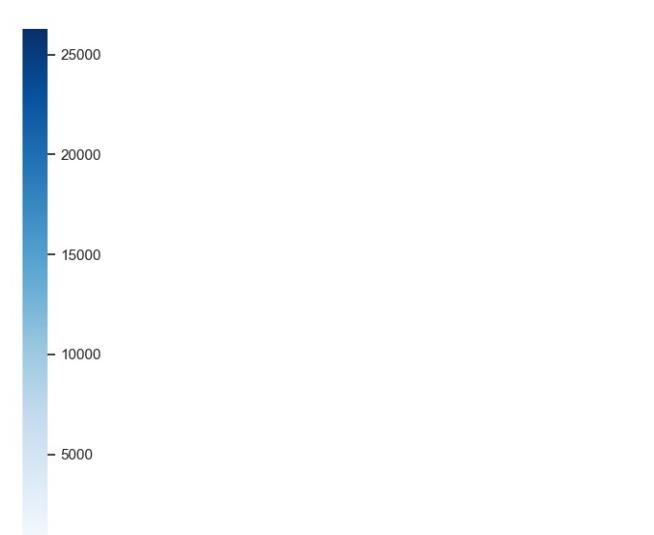
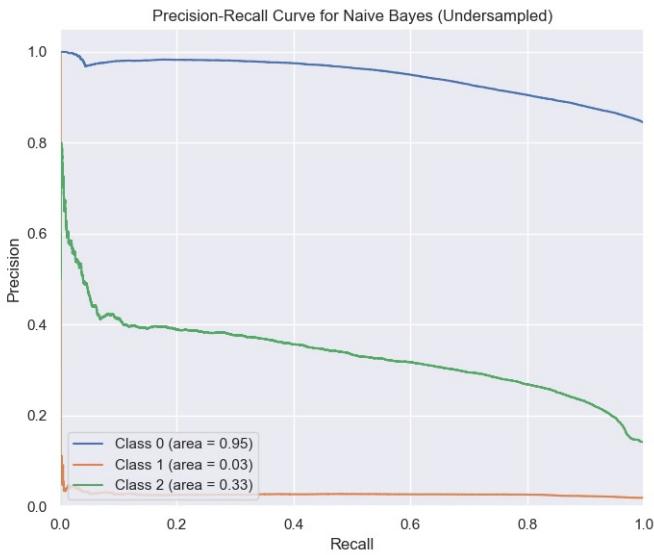
```
precision    recall   f1-score   support
0.0       0.946803  0.613857  0.744815  42795.000000
1.0       0.026742  0.316737  0.049320  944.000000
2.0       0.324244  0.547235  0.407210  6997.000000
accuracy   0.599141  0.599141  0.599141  0.599141
macro avg   0.432596  0.492610  0.400448  50736.000000
weighted avg 0.843827  0.599141  0.685316  50736.000000
```



Naive Bayes Model Evaluation

Naive Bayes (Original Data)

- Accuracy: 75.37%
- Precision, Recall, and F1-Score:
 - No Diabetes (0): Precision = 0.91, Recall = 0.80, F1-Score = 0.85
 - Prediabetes (1): Precision = 0.04, Recall = 0.02, F1-Score = 0.03
 - Diabetes (2): Precision = 0.32, Recall = 0.56, F1-Score = 0.40
- Confusion Matrix Analysis:
 - The majority of predictions are correct for the "No Diabetes" class.



- Significant misclassifications between "No Diabetes" and "Diabetes" and poor performance for "Prediabetes".
- **ROC and Precision-Recall Curves:**
 - ROC-AUC scores: 0.78 (Class 0), 0.65 (Class 1), 0.79 (Class 2)
 - Precision-Recall curves show high precision for Class 0 but low for Classes 1 and 2.

Naive Bayes (Oversampled Data)

- **Accuracy:** 58.61%
- **Precision, Recall, and F1-Score:**
 - **No Diabetes (0):** Precision = 0.95, Recall = 0.59, F1-Score = 0.73
 - **Prediabetes (1):** Precision = 0.03, Recall = 0.29, F1-Score = 0.05
 - **Diabetes (2):** Precision = 0.32, Recall = 0.61, F1-Score = 0.42
- **Confusion Matrix Analysis:**
 - More correct predictions for "No Diabetes", but significant misclassifications.
 - Improved recall for "Prediabetes" but still low precision.
- **ROC and Precision-Recall Curves:**
 - ROC-AUC scores: 0.77 (Class 0), 0.58 (Class 1), 0.78 (Class 2)
 - Precision-Recall curves show a trade-off between precision and recall for Classes 1 and 2.

Naive Bayes (Undersampled Data)

- **Accuracy:** 67.68%
- **Precision, Recall, and F1-Score:**
 - **No Diabetes (0):** Precision = 0.93, Recall = 0.70, F1-Score = 0.80
 - **Prediabetes (1):** Precision = 0.03, Recall = 0.15, F1-Score = 0.05
 - **Diabetes (2):** Precision = 0.32, Recall = 0.60, F1-Score = 0.42
- **Confusion Matrix Analysis:**
 - Better performance for "No Diabetes" compared to the oversampled data.
 - Significant misclassifications in "Prediabetes" and "Diabetes".
- **ROC and Precision-Recall Curves:**
 - ROC-AUC scores: 0.78 (Class 0), 0.63 (Class 1), 0.78 (Class 2)
 - Precision-Recall curves show high precision for Class 0 but low for Classes 1 and 2.

Naive Bayes (SMOTEENN Data)

- **Accuracy:** 59.91%
- **Precision, Recall, and F1-Score:**
 - **No Diabetes (0):** Precision = 0.95, Recall = 0.61, F1-Score = 0.74
 - **Prediabetes (1):** Precision = 0.03, Recall = 0.32, F1-Score = 0.05
 - **Diabetes (2):** Precision = 0.32, Recall = 0.55, F1-Score = 0.41
- **Confusion Matrix Analysis:**
 - Similar performance to the undersampled data.
 - More misclassifications in "Prediabetes" and "Diabetes".
- **ROC and Precision-Recall Curves:**
 - ROC-AUC scores: 0.77 (Class 0), 0.61 (Class 1), 0.78 (Class 2)
 - Precision-Recall curves show high precision for Class 0 but low for Classes 1 and 2.

General Insights

1. **Accuracy vs. Recall Trade-off:**
 - The original Naive Bayes model has higher overall accuracy but lower recall for "Prediabetes" and "Diabetes".
 - The oversampled and SMOTEENN models have better recall for these classes, making them more suitable for identifying more instances of these conditions despite lower overall accuracy.
2. **Class Imbalance Challenge:**
 - All models struggle with the "Prediabetes" class due to its small sample size. Further data collection or different modeling approaches may be needed to improve predictions for this class.
3. **Business Implications:**
 - **Original Model:** Higher accuracy can lead to better general performance in identifying non-diabetic individuals, reducing unnecessary interventions.
 - **Oversampled and SMOTEENN Models:** Improved recall for "Diabetes" suggests better early detection, potentially leading to timely intervention and better patient outcomes despite a higher rate of false positives.
4. **Model Selection:**
 - The choice between the models depends on the specific needs of the healthcare setting. If the priority is to minimize missed diabetes cases, the oversampled or SMOTEENN models might be preferred. For a balanced approach with high recall and precision, the SMOTEENN model appears most promising.

KNN

```
# In [56]: # KNN + Original Data
accuracy_original, report_original, y_pred_original = train_and_evaluate_knn(X_train, X_test, y_train, y_test)
display_results(accuracy_original, report_original, y_test, y_pred_original, "KNN (Original)")
plot_roc_pr_curves(y_test, y_pred_original, "KNN (Original)")

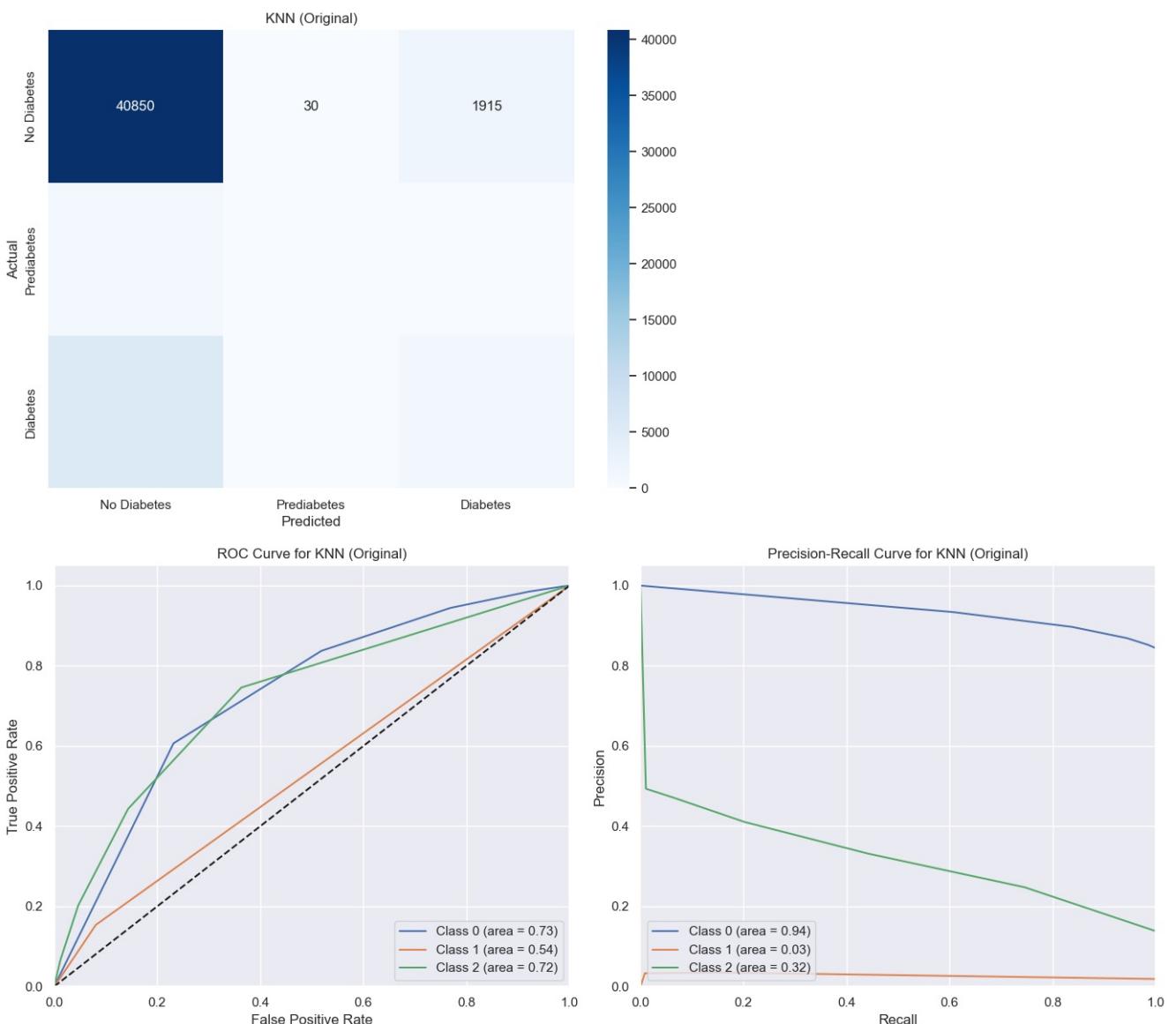
# KNN + Oversampled Data
accuracy_oversampled, report_oversampled, y_pred_oversampled = train_and_evaluate_knn(X_oversampled, X_test, y_oversampled, y_test)
display_results(accuracy_oversampled, report_oversampled, y_test, y_pred_oversampled, "KNN (Oversampled)")
plot_roc_pr_curves(y_test, y_pred_oversampled, "KNN (Oversampled)")

# KNN + Undersampled Data
accuracy_undersampled, report_undersampled, y_pred_undersampled, y_prob_undersampled = train_and_evaluate_knn(X_undersampled, X_test, y_undersampled, y_test)
display_results(accuracy_undersampled, report_undersampled, y_test, y_pred_undersampled, "KNN (Undersampled)")
plot_roc_pr_curves(y_test, y_prob_undersampled, "KNN (Undersampled)")

# KNN + SMOTEENN Data
accuracy_smoteenn, report_smoteenn, y_pred_smoteenn, y_prob_smoteenn = train_and_evaluate_knn(X_smoteenn, X_test, y_smoteenn, y_test)
display_results(accuracy_smoteenn, report_smoteenn, y_test, y_pred_smoteenn, "KNN (SMOTEENN)")
plot_roc_pr_curves(y_test, y_prob_smoteenn, "KNN (SMOTEENN)")

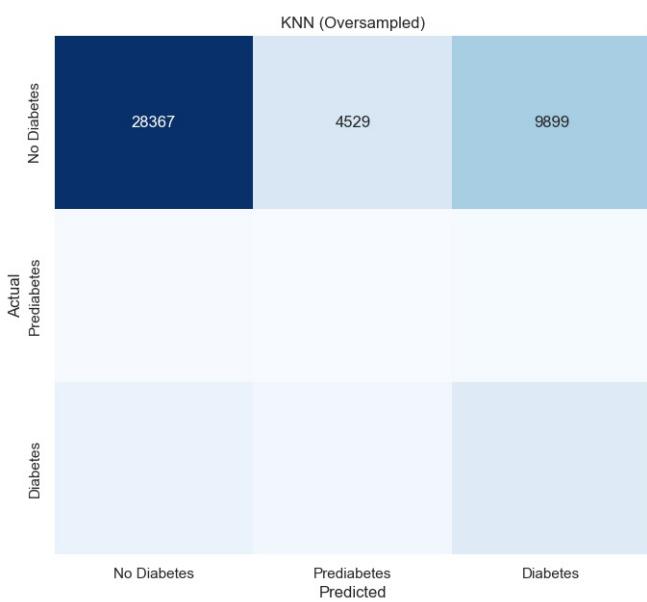
KNN (Original)
Accuracy: 0.8330376852727847
```

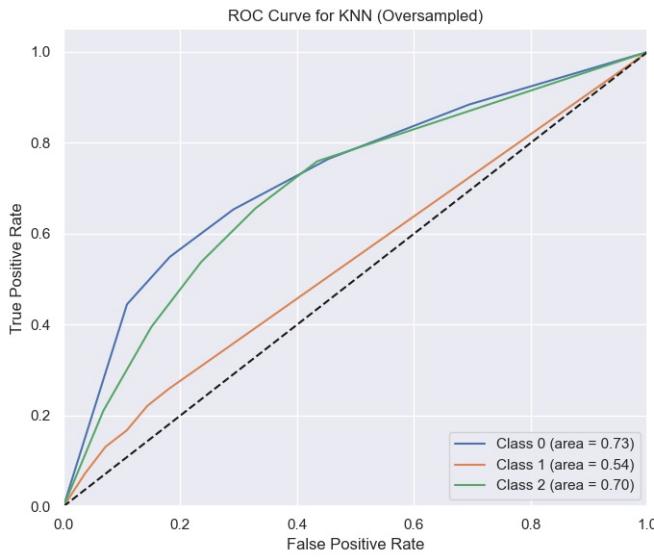
	precision	recall	f1-score	support
0.0	0.864752	0.954551	0.907435	42795.000000
1.0	0.000000	0.000000	0.000000	944.000000
2.0	0.410145	0.202230	0.270891	6997.000000
accuracy	0.833038	0.833038	0.833038	
macro avg	0.424966	0.385593	0.392775	50736.000000
weighted avg	0.785967	0.833038	0.802765	50736.000000



KNN (Oversampled)
Accuracy: 0.636648535856197

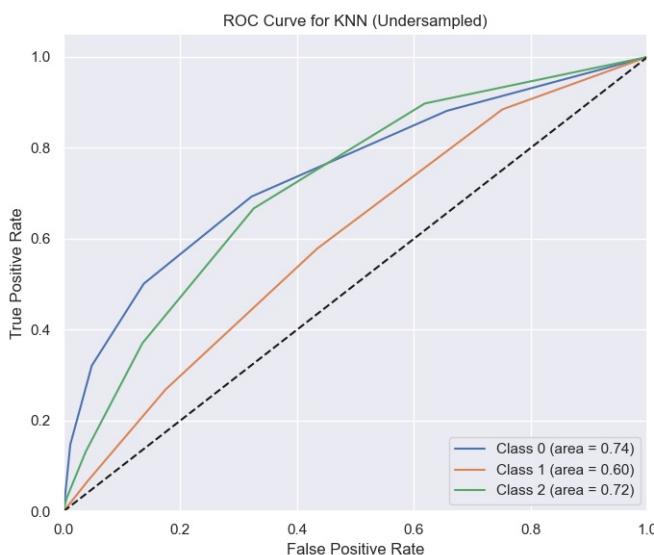
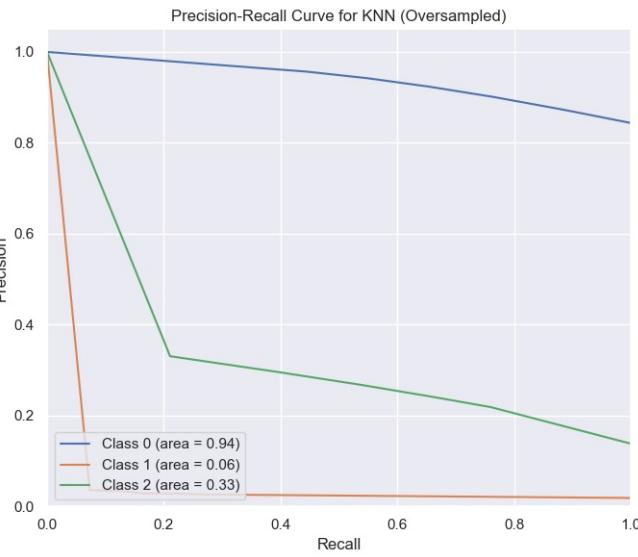
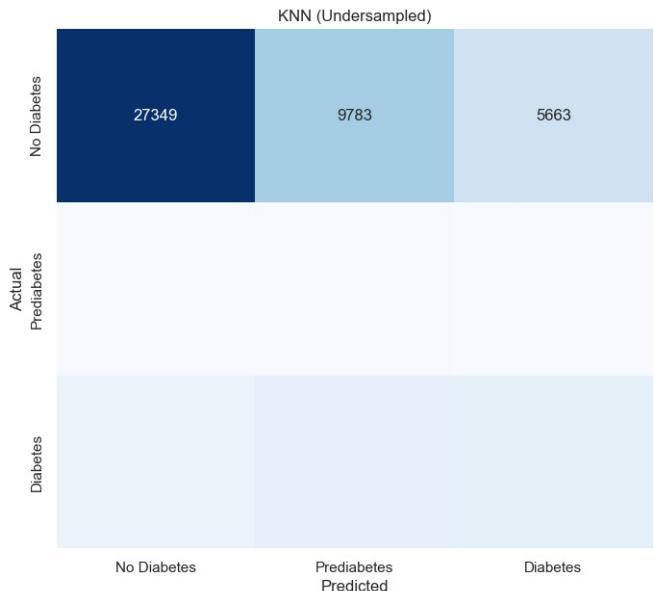
```
precision    recall   f1-score   support
0.0      0.920319  0.662858  0.770654  42795.000000
1.0      0.029563  0.183263  0.050912  944.000000
2.0      0.267477  0.537516  0.357204  6997.000000
accuracy          0.636649  0.636649  0.636649
macro avg      0.405786  0.461212  0.392923  50736.000000
weighted avg     0.813712  0.636649  0.700243  50736.000000
```





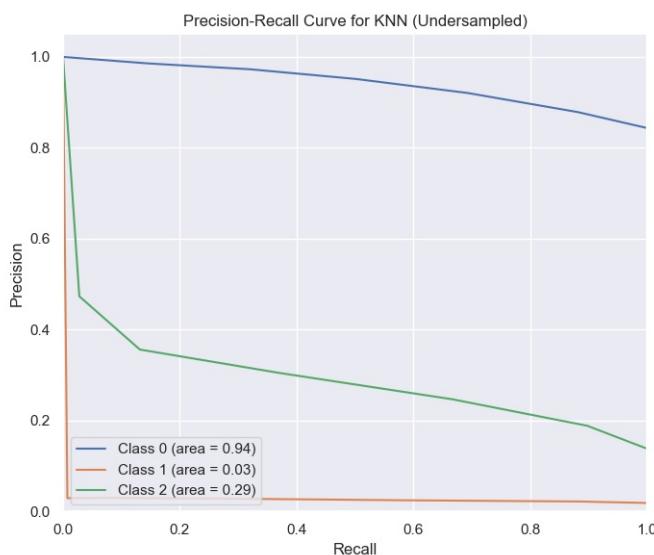
KNN (Undersampled)
Accuracy: 0.5970908230842006

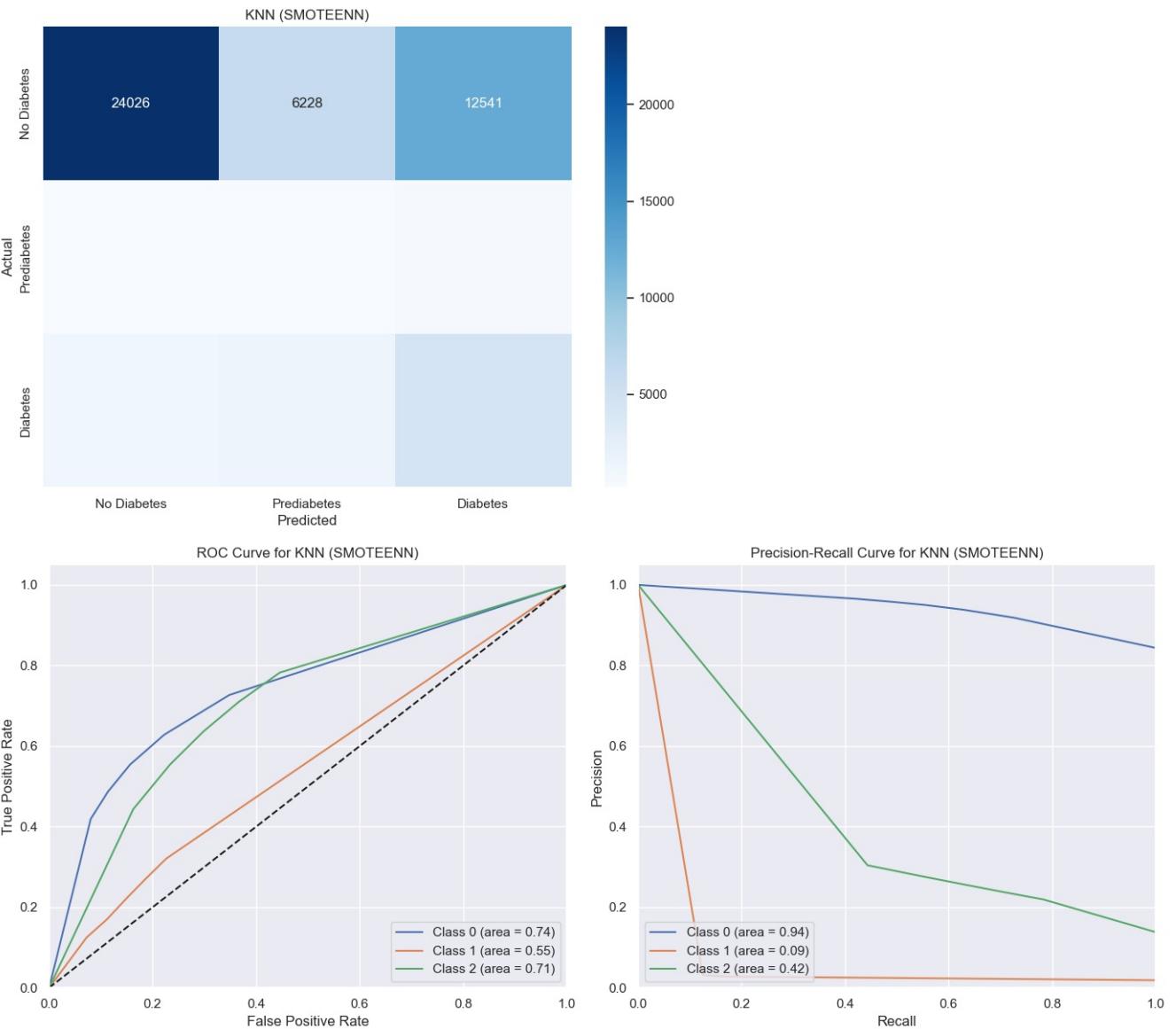
```
precision    recall  f1-score   support
0.0       0.926237  0.639070  0.756312  42795.000000
1.0       0.027931  0.376059  0.051999  944.000000
2.0       0.304742  0.370159  0.334280  6997.000000
accuracy   0.597091  0.597091  0.597091  0.597091
macro avg   0.419637  0.461763  0.380864  50736.000000
weighted avg 0.823813  0.597091  0.685005  50736.000000
```



KNN (SMOTEENN)
Accuracy: 0.5654367707347839

```
precision    recall  f1-score   support
0.0       0.948145  0.561421  0.705247  42795.000000
1.0       0.026665  0.223517  0.047646  944.000000
2.0       0.254590  0.636130  0.363644  6997.000000
accuracy   0.565437  0.565437  0.565437  0.565437
macro avg   0.409800  0.473689  0.372178  50736.000000
weighted avg 0.835352  0.565437  0.645901  50736.000000
```





Neural Networks (W/ RandomForest for Feature Selection)

Neural networks are chosen because of their flexibility and ability to model complex, non-linear relationships, even though the data might exhibit linear interactions. This choice ensures the model can capture any subtle non-linear patterns that r

```
In [58]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input, Dropout
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import LearningRateScheduler, EarlyStopping
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score, roc_curve, auc, precision_recall_curve, accuracy_score
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
from imblearn.over_sampling import SMOTE

# Load the data
file_path = "diabetes_012_health_indicators_BRFSS2015.csv"
data = pd.read_csv(file_path)

# Define the features and target
X = data.drop("Diabetes_012", axis=1)
y = data["Diabetes_012"]

# Normalize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Use RandomForest to determine feature importance
rf = RandomForestClassifier(random_state=42)
rf.fit(X_scaled, y)
importances = rf.feature_importances_

# Select features based on importance
model_selector = SelectFromModel(rf, prefit=True, threshold='mean')
X_important = model_selector.transform(X_scaled)

# Apply SMOTE to handle class imbalance
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_important, y)

# Convert target to categorical
y_categorical = to_categorical(y_resampled)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_categorical, test_size=0.3, random_state=42)

# Print the shape of the training and testing data
print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")

# Define the neural network model
model = Sequential()
model.add(Input(shape=(X_train.shape[1],)))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(3, activation='softmax'))
```

```

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test))

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_test_classes = np.argmax(y_test, axis=1)

report = classification_report(y_test_classes, y_pred_classes)
title = 'Neural Network Model Evaluation'
display_results(accuracy, report, y_test_classes, y_pred_classes, title)

# Combined Accuracy and Loss Plot
fig, ax1 = plt.subplots(figsize=(12, 6))

ax1.set_xlabel('Epoch')
ax1.set_ylabel('Accuracy')
ax1.plot(history.history['accuracy'], label='Train Accuracy', color='tab:blue')
ax1.plot(history.history['val_accuracy'], label='Validation Accuracy', color='tab:orange')
ax1.tick_params(axis='Y')
ax1.legend(loc='upper left')

ax2 = ax1.twinx() # instantiate a second axes that shares the same x-axis
ax2.set_ylabel('Loss')
ax2.plot(history.history['loss'], label='Train Loss', color='tab:green')
ax2.plot(history.history['val_loss'], label='Validation Loss', color='tab:red')
ax2.tick_params(axis='Y')
ax2.legend(loc='upper right')

fig.tight_layout() # otherwise the right y-label is slightly clipped
plt.title('Model Accuracy and Loss')
plt.show()

# Calculate ROC curve and ROC AUC
fpr, tpr, _ = roc_curve(y_test.ravel(), y_pred.ravel())
roc_auc = auc(fpr, tpr)

# Calculate Precision-Recall curve
precision, recall, _ = precision_recall_curve(y_test.ravel(), y_pred.ravel())

# Combined ROC and Precision-Recall Plot
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

# ROC Curve
ax1.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
ax1.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
ax1.set_xlim([0.0, 1.0])
ax1.set_ylim([0.0, 1.05])
ax1.set_xlabel('False Positive Rate')
ax1.set_ylabel('True Positive Rate')
ax1.set_title('Receiver Operating Characteristic (ROC)')
ax1.legend(loc='lower right')

# Precision-Recall Curve
ax2.plot(recall, precision, color='blue', lw=2)
ax2.set_xlabel('Recall')
ax2.set_ylabel('Precision')
ax2.set_title('Precision-Recall Curve')

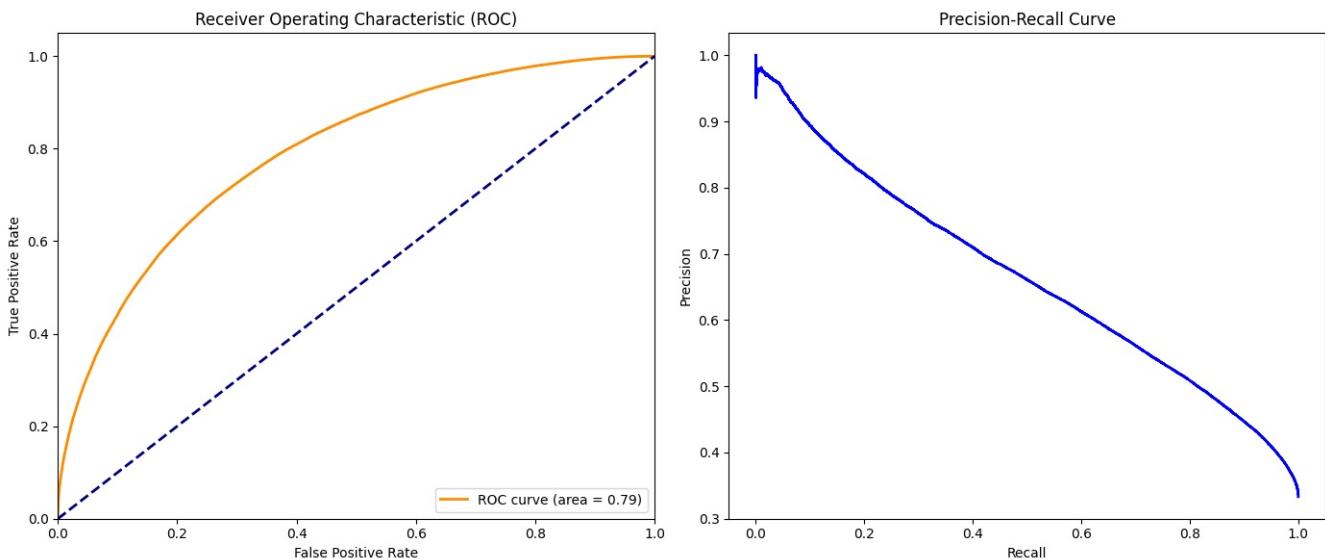
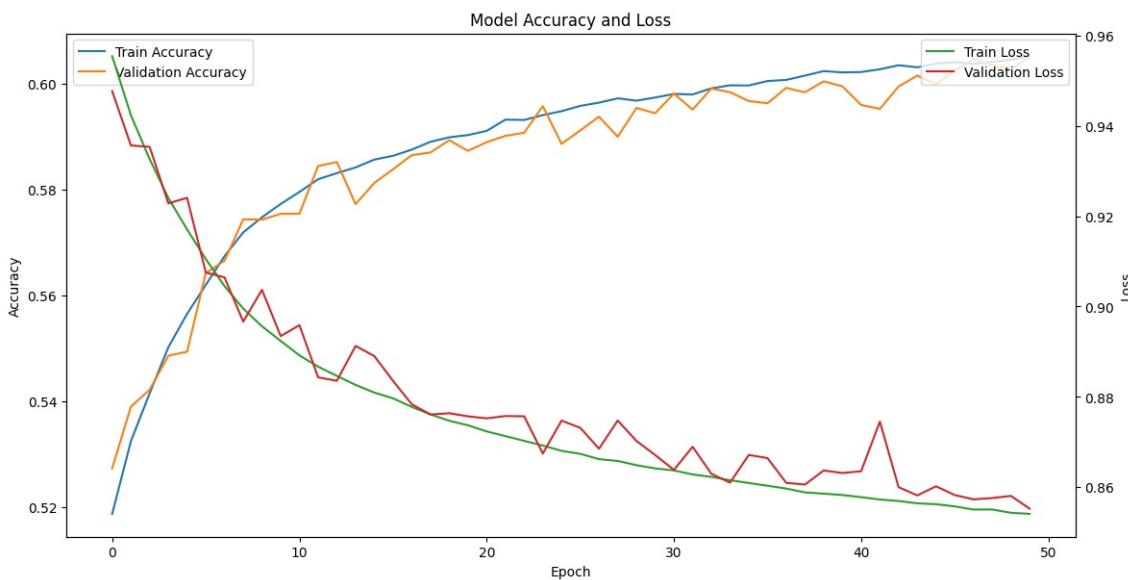
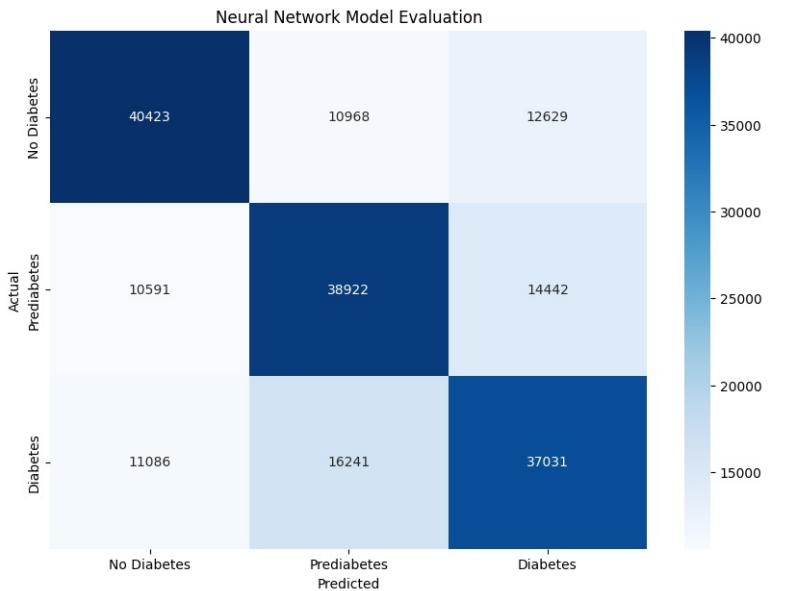
plt.tight_layout()
plt.show()

```

X_train shape: (448776, 7)
X_test shape: (192333, 7)
Epoch 1/50
14025/14025 6s 409us/step - accuracy: 0.5109 - loss: 0.9628 - val_accuracy: 0.5273 - val_loss: 0.9477
Epoch 2/50
14025/14025 6s 410us/step - accuracy: 0.5310 - loss: 0.9445 - val_accuracy: 0.5390 - val_loss: 0.9357
Epoch 3/50
14025/14025 6s 417us/step - accuracy: 0.5383 - loss: 0.9351 - val_accuracy: 0.5421 - val_loss: 0.9353
Epoch 4/50
14025/14025 6s 439us/step - accuracy: 0.5487 - loss: 0.9250 - val_accuracy: 0.5486 - val_loss: 0.9228
Epoch 5/50
14025/14025 6s 422us/step - accuracy: 0.5545 - loss: 0.9189 - val_accuracy: 0.5493 - val_loss: 0.9241
Epoch 6/50
14025/14025 6s 434us/step - accuracy: 0.5614 - loss: 0.9128 - val_accuracy: 0.5642 - val_loss: 0.9075
Epoch 7/50
14025/14025 6s 445us/step - accuracy: 0.5664 - loss: 0.9058 - val_accuracy: 0.5665 - val_loss: 0.9064
Epoch 8/50
14025/14025 6s 441us/step - accuracy: 0.5715 - loss: 0.9003 - val_accuracy: 0.5743 - val_loss: 0.8966
Epoch 9/50
14025/14025 6s 457us/step - accuracy: 0.5749 - loss: 0.8953 - val_accuracy: 0.5743 - val_loss: 0.9037
Epoch 10/50
14025/14025 6s 457us/step - accuracy: 0.5782 - loss: 0.8920 - val_accuracy: 0.5754 - val_loss: 0.8934
Epoch 11/50
14025/14025 7s 469us/step - accuracy: 0.5799 - loss: 0.8898 - val_accuracy: 0.5754 - val_loss: 0.8959
Epoch 12/50
14025/14025 7s 499us/step - accuracy: 0.5828 - loss: 0.8867 - val_accuracy: 0.5844 - val_loss: 0.8843
Epoch 13/50
14025/14025 7s 478us/step - accuracy: 0.5824 - loss: 0.8845 - val_accuracy: 0.5852 - val_loss: 0.8835
Epoch 14/50
14025/14025 8s 539us/step - accuracy: 0.5842 - loss: 0.8826 - val_accuracy: 0.5772 - val_loss: 0.8913
Epoch 15/50
14025/14025 7s 485us/step - accuracy: 0.5850 - loss: 0.8811 - val_accuracy: 0.5812 - val_loss: 0.8890
Epoch 16/50
14025/14025 7s 481us/step - accuracy: 0.5866 - loss: 0.8801 - val_accuracy: 0.5838 - val_loss: 0.8835
Epoch 17/50
14025/14025 6s 424us/step - accuracy: 0.5881 - loss: 0.8768 - val_accuracy: 0.5865 - val_loss: 0.8783
Epoch 18/50
14025/14025 6s 422us/step - accuracy: 0.5890 - loss: 0.8762 - val_accuracy: 0.5870 - val_loss: 0.8761
Epoch 19/50
14025/14025 6s 427us/step - accuracy: 0.5892 - loss: 0.8743 - val_accuracy: 0.5893 - val_loss: 0.8764
Epoch 20/50
14025/14025 6s 422us/step - accuracy: 0.5897 - loss: 0.8750 - val_accuracy: 0.5873 - val_loss: 0.8757
Epoch 21/50
14025/14025 6s 435us/step - accuracy: 0.5900 - loss: 0.8737 - val_accuracy: 0.5889 - val_loss: 0.8752
Epoch 22/50
14025/14025 6s 413us/step - accuracy: 0.5935 - loss: 0.8703 - val_accuracy: 0.5901 - val_loss: 0.8757
Epoch 23/50
14025/14025 6s 420us/step - accuracy: 0.5920 - loss: 0.8713 - val_accuracy: 0.5907 - val_loss: 0.8757
Epoch 24/50
14025/14025 6s 415us/step - accuracy: 0.5943 - loss: 0.8689 - val_accuracy: 0.5957 - val_loss: 0.8674
Epoch 25/50
14025/14025 6s 425us/step - accuracy: 0.5951 - loss: 0.8684 - val_accuracy: 0.5886 - val_loss: 0.8747
Epoch 26/50
14025/14025 6s 458us/step - accuracy: 0.5964 - loss: 0.8668 - val_accuracy: 0.5911 - val_loss: 0.8731
Epoch 27/50
14025/14025 6s 429us/step - accuracy: 0.5970 - loss: 0.8658 - val_accuracy: 0.5937 - val_loss: 0.8685
Epoch 28/50
14025/14025 6s 413us/step - accuracy: 0.5969 - loss: 0.8660 - val_accuracy: 0.5899 - val_loss: 0.8747
Epoch 29/50
14025/14025 6s 410us/step - accuracy: 0.5973 - loss: 0.8648 - val_accuracy: 0.5954 - val_loss: 0.8702
Epoch 30/50
14025/14025 6s 418us/step - accuracy: 0.5970 - loss: 0.8645 - val_accuracy: 0.5944 - val_loss: 0.8671
Epoch 31/50
14025/14025 6s 446us/step - accuracy: 0.5989 - loss: 0.8627 - val_accuracy: 0.5981 - val_loss: 0.8638
Epoch 32/50
14025/14025 6s 430us/step - accuracy: 0.5977 - loss: 0.8620 - val_accuracy: 0.5951 - val_loss: 0.8689
Epoch 33/50
14025/14025 6s 424us/step - accuracy: 0.5977 - loss: 0.8634 - val_accuracy: 0.5991 - val_loss: 0.8630
Epoch 34/50
14025/14025 7s 485us/step - accuracy: 0.5993 - loss: 0.8619 - val_accuracy: 0.5984 - val_loss: 0.8610
Epoch 35/50
14025/14025 6s 423us/step - accuracy: 0.5999 - loss: 0.8596 - val_accuracy: 0.5967 - val_loss: 0.8671
Epoch 36/50
14025/14025 6s 401us/step - accuracy: 0.5986 - loss: 0.8624 - val_accuracy: 0.5963 - val_loss: 0.8664
Epoch 37/50
14025/14025 6s 405us/step - accuracy: 0.6012 - loss: 0.8595 - val_accuracy: 0.5992 - val_loss: 0.8609
Epoch 38/50
14025/14025 6s 418us/step - accuracy: 0.6017 - loss: 0.8586 - val_accuracy: 0.5983 - val_loss: 0.8606
Epoch 39/50
14025/14025 6s 428us/step - accuracy: 0.6030 - loss: 0.8580 - val_accuracy: 0.6004 - val_loss: 0.8637
Epoch 40/50
14025/14025 6s 415us/step - accuracy: 0.6015 - loss: 0.8602 - val_accuracy: 0.5995 - val_loss: 0.8631
Epoch 41/50
14025/14025 6s 420us/step - accuracy: 0.6024 - loss: 0.8578 - val_accuracy: 0.5960 - val_loss: 0.8635
Epoch 42/50
14025/14025 6s 414us/step - accuracy: 0.6037 - loss: 0.8562 - val_accuracy: 0.5952 - val_loss: 0.8745
Epoch 43/50
14025/14025 6s 414us/step - accuracy: 0.6046 - loss: 0.8566 - val_accuracy: 0.5994 - val_loss: 0.8600
Epoch 44/50
14025/14025 6s 415us/step - accuracy: 0.6035 - loss: 0.8565 - val_accuracy: 0.6015 - val_loss: 0.8582
Epoch 45/50
14025/14025 6s 418us/step - accuracy: 0.6046 - loss: 0.8548 - val_accuracy: 0.5998 - val_loss: 0.8602
Epoch 46/50
14025/14025 6s 426us/step - accuracy: 0.6059 - loss: 0.8535 - val_accuracy: 0.6025 - val_loss: 0.8582
Epoch 47/50
14025/14025 6s 403us/step - accuracy: 0.6045 - loss: 0.8550 - val_accuracy: 0.6046 - val_loss: 0.8573
Epoch 48/50
14025/14025 6s 409us/step - accuracy: 0.6034 - loss: 0.8556 - val_accuracy: 0.6035 - val_loss: 0.8576
Epoch 49/50
14025/14025 6s 393us/step - accuracy: 0.6046 - loss: 0.8549 - val_accuracy: 0.6027 - val_loss: 0.8581
Epoch 50/50
14025/14025 5s 390us/step - accuracy: 0.6062 - loss: 0.8527 - val_accuracy: 0.6051 - val_loss: 0.8552
6011/6011 1s 238us/step - accuracy: 0.6053 - loss: 0.8559
6011/6011 1s 224us/step

Neural Network Model Evaluation
Accuracy: 0.6050755977630615

	precision	recall	f1-score	support
0	0.65	0.63	0.64	64020
1	0.59	0.61	0.60	63955
2	0.58	0.58	0.58	64358
accuracy		0.61	0.62	192333
macro avg	0.61	0.61	0.61	192333
weighted avg	0.61	0.61	0.61	192333



Result Interpretation

Loss:

The loss value is not provided in this summary, but it typically measures the error in the model's predictions. Lower values indicate better performance.

Accuracy:

This value indicates that 61% of the model's predictions on the test set are correct. Accuracy is the ratio of correctly predicted observations to the total observations.

Precision:

Precision indicates the proportion of positive identifications that were actually correct for each class. Higher precision means fewer false positives. Class 0 has the highest precision, while Class 2 has the lowest.

Recall:

Recall measures the proportion of actual positives that were correctly identified by the model for each class. Higher recall means fewer false negatives. Class 1 has the highest recall, while Class 2 has the lowest.

F1 Score:

The F1 score is the harmonic mean of precision and recall, providing a single metric that balances both concerns for each class. Class 0 has the highest F1 score, while Class 2 has the lowest.

Support:

Support is the number of actual occurrences of each class in the dataset. It provides context to the precision, recall, and F1 score metrics.

Macro Average:

The macro average calculates the average of each metric by treating all classes equally, providing an overall performance metric without considering class imbalance.

Weighted Average:

The weighted average calculates the average of each metric by considering the support (the number of true instances for each class), providing a more balanced overall performance metric that accounts for class imbalance.

Summary

The model shows moderate performance with balanced precision, recall, and F1 scores across the classes. However, the accuracy of 61% indicates that there is room for improvement. The relatively close values of precision, recall, and F1 scores, but the model's ability to distinguish between classes could be enhanced further.

Adding Dropout Layers to Prevent Overfitting

Dropout layers are being added to the neural network to randomly deactivate a portion of neurons during training, which helps prevent the model from becoming too reliant on specific neurons. This technique reduces overfitting by promoting a more balanced performance on unseen data.

```
In [9]: # Define the neural network model with dropout layers
model = Sequential()
model.add(Input(shape=(X_train.shape[1],)))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(3, activation='softmax'))

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=[accuracy])

# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test))

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_test_classes = np.argmax(y_test, axis=1)

report = classification_report(y_test_classes, y_pred_classes)
title = 'Neural Network Model Evaluation w/ Dropout Layers to Prevent Overfitting'
display_results(accuracy, report, y_test_classes, y_pred_classes, title)

# Combined Accuracy and Loss Plot
fig, ax1 = plt.subplots(figsize=(12, 6))

ax1.set_xlabel('Epoch')
ax1.set_ylabel('Accuracy')
ax1.plot(history.history['accuracy'], label='Train Accuracy', color='tab:blue')
ax1.plot(history.history['val_accuracy'], label='Validation Accuracy', color='tab:orange')
ax1.tick_params(axis='y')
ax1.legend(loc='upper left')

ax2 = ax1.twinx() # instantiate a second axes that shares the same x-axis
ax2.set_ylabel('Loss')
ax2.plot(history.history['loss'], label='Train Loss', color='tab:green')
ax2.plot(history.history['val_loss'], label='Validation Loss', color='tab:red')
ax2.tick_params(axis='y')
ax2.legend(loc='upper right')

fig.tight_layout() # otherwise the right y-label is slightly clipped
plt.title('Model Accuracy and Loss')
plt.show()

# Calculate ROC curve and ROC AUC
fpr, tpr, _ = roc_curve(y_test.ravel(), y_pred.ravel())
roc_auc = auc(fpr, tpr)

# Calculate Precision-Recall curve
precision, recall, _ = precision_recall_curve(y_test.ravel(), y_pred.ravel())

# Combined ROC and Precision-Recall Plot
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

# ROC Curve
ax1.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
ax1.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
ax1.set_xlim([0.0, 1.0])
ax1.set_ylim([0.0, 1.05])
ax1.set_xlabel('False Positive Rate')
ax1.set_ylabel('True Positive Rate')
ax1.set_title('Receiver Operating Characteristic (ROC)')
ax1.legend(loc='lower right')

# Precision-Recall Curve
ax2.plot(recall, precision, color='blue', lw=2)
ax2.set_xlabel('Recall')
ax2.set_ylabel('Precision')
ax2.set_title('Precision-Recall Curve')

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

Epoch 1/50
14025/14025 6s 422us/step - accuracy: 0.4864 - loss: 0.9983 - val_accuracy: 0.5116 - val_loss: 0.9647
 Epoch 2/50
14025/14025 6s 416us/step - accuracy: 0.5049 - loss: 0.9754 - val_accuracy: 0.5122 - val_loss: 0.9617
 Epoch 3/50
14025/14025 6s 411us/step - accuracy: 0.5069 - loss: 0.9748 - val_accuracy: 0.5143 - val_loss: 0.9590
 Epoch 4/50
14025/14025 6s 413us/step - accuracy: 0.5086 - loss: 0.9742 - val_accuracy: 0.5139 - val_loss: 0.9598
 Epoch 5/50
14025/14025 6s 414us/step - accuracy: 0.5074 - loss: 0.9736 - val_accuracy: 0.5159 - val_loss: 0.9571
 Epoch 6/50
14025/14025 6s 413us/step - accuracy: 0.5077 - loss: 0.9734 - val_accuracy: 0.5156 - val_loss: 0.9571
 Epoch 7/50
14025/14025 6s 412us/step - accuracy: 0.5091 - loss: 0.9726 - val_accuracy: 0.5179 - val_loss: 0.9566
 Epoch 8/50
14025/14025 6s 410us/step - accuracy: 0.5086 - loss: 0.9719 - val_accuracy: 0.5145 - val_loss: 0.9597
 Epoch 9/50
14025/14025 6s 414us/step - accuracy: 0.5094 - loss: 0.9712 - val_accuracy: 0.5180 - val_loss: 0.9567
 Epoch 10/50
14025/14025 6s 417us/step - accuracy: 0.5109 - loss: 0.9696 - val_accuracy: 0.5148 - val_loss: 0.9562
 Epoch 11/50
14025/14025 6s 410us/step - accuracy: 0.5118 - loss: 0.9695 - val_accuracy: 0.5178 - val_loss: 0.9559
 Epoch 12/50
14025/14025 6s 434us/step - accuracy: 0.5107 - loss: 0.9704 - val_accuracy: 0.5174 - val_loss: 0.9561
 Epoch 13/50
14025/14025 6s 412us/step - accuracy: 0.5097 - loss: 0.9702 - val_accuracy: 0.5196 - val_loss: 0.9556
 Epoch 14/50
14025/14025 6s 414us/step - accuracy: 0.5097 - loss: 0.9712 - val_accuracy: 0.5180 - val_loss: 0.9540
 Epoch 15/50
14025/14025 6s 456us/step - accuracy: 0.5098 - loss: 0.9712 - val_accuracy: 0.5190 - val_loss: 0.9544
 Epoch 16/50
14025/14025 6s 422us/step - accuracy: 0.5094 - loss: 0.9708 - val_accuracy: 0.5199 - val_loss: 0.9533
 Epoch 17/50
14025/14025 6s 430us/step - accuracy: 0.5102 - loss: 0.9697 - val_accuracy: 0.5184 - val_loss: 0.9548
 Epoch 18/50
14025/14025 6s 428us/step - accuracy: 0.5108 - loss: 0.9710 - val_accuracy: 0.5171 - val_loss: 0.9542
 Epoch 19/50
14025/14025 6s 416us/step - accuracy: 0.5099 - loss: 0.9707 - val_accuracy: 0.5200 - val_loss: 0.9593
 Epoch 20/50
14025/14025 6s 416us/step - accuracy: 0.5137 - loss: 0.9694 - val_accuracy: 0.5165 - val_loss: 0.9549
 Epoch 21/50
14025/14025 6s 416us/step - accuracy: 0.5117 - loss: 0.9681 - val_accuracy: 0.5172 - val_loss: 0.9539
 Epoch 22/50
14025/14025 6s 439us/step - accuracy: 0.5119 - loss: 0.9699 - val_accuracy: 0.5204 - val_loss: 0.9555
 Epoch 23/50
14025/14025 6s 419us/step - accuracy: 0.5120 - loss: 0.9701 - val_accuracy: 0.5190 - val_loss: 0.9537
 Epoch 24/50
14025/14025 6s 418us/step - accuracy: 0.5125 - loss: 0.9687 - val_accuracy: 0.5185 - val_loss: 0.9557
 Epoch 25/50
14025/14025 6s 413us/step - accuracy: 0.5123 - loss: 0.9689 - val_accuracy: 0.5209 - val_loss: 0.9527
 Epoch 26/50
14025/14025 6s 415us/step - accuracy: 0.5107 - loss: 0.9702 - val_accuracy: 0.5182 - val_loss: 0.9566
 Epoch 27/50
14025/14025 6s 416us/step - accuracy: 0.5105 - loss: 0.9693 - val_accuracy: 0.5169 - val_loss: 0.9531
 Epoch 28/50
14025/14025 6s 418us/step - accuracy: 0.5090 - loss: 0.9700 - val_accuracy: 0.5199 - val_loss: 0.9562
 Epoch 29/50
14025/14025 6s 416us/step - accuracy: 0.5103 - loss: 0.9690 - val_accuracy: 0.5156 - val_loss: 0.9532
 Epoch 30/50
14025/14025 6s 417us/step - accuracy: 0.5089 - loss: 0.9705 - val_accuracy: 0.5178 - val_loss: 0.9534
 Epoch 31/50
14025/14025 6s 416us/step - accuracy: 0.5119 - loss: 0.9696 - val_accuracy: 0.5184 - val_loss: 0.9544
 Epoch 32/50
14025/14025 6s 417us/step - accuracy: 0.5126 - loss: 0.9685 - val_accuracy: 0.5181 - val_loss: 0.9564
 Epoch 33/50
14025/14025 6s 432us/step - accuracy: 0.5122 - loss: 0.9698 - val_accuracy: 0.5208 - val_loss: 0.9545
 Epoch 34/50
14025/14025 6s 416us/step - accuracy: 0.5107 - loss: 0.9691 - val_accuracy: 0.5206 - val_loss: 0.9541
 Epoch 35/50
14025/14025 6s 462us/step - accuracy: 0.5102 - loss: 0.9695 - val_accuracy: 0.5210 - val_loss: 0.9527
 Epoch 36/50
14025/14025 6s 435us/step - accuracy: 0.5124 - loss: 0.9684 - val_accuracy: 0.5196 - val_loss: 0.9531
 Epoch 37/50
14025/14025 6s 435us/step - accuracy: 0.5115 - loss: 0.9683 - val_accuracy: 0.5185 - val_loss: 0.9540
 Epoch 38/50
14025/14025 6s 427us/step - accuracy: 0.5114 - loss: 0.9685 - val_accuracy: 0.5195 - val_loss: 0.9524
 Epoch 39/50
14025/14025 6s 427us/step - accuracy: 0.5115 - loss: 0.9680 - val_accuracy: 0.5197 - val_loss: 0.9539
 Epoch 40/50
14025/14025 6s 426us/step - accuracy: 0.5088 - loss: 0.9693 - val_accuracy: 0.5194 - val_loss: 0.9569
 Epoch 41/50
14025/14025 6s 421us/step - accuracy: 0.5114 - loss: 0.9692 - val_accuracy: 0.5207 - val_loss: 0.9542
 Epoch 42/50
14025/14025 6s 421us/step - accuracy: 0.5116 - loss: 0.9687 - val_accuracy: 0.5182 - val_loss: 0.9536
 Epoch 43/50
14025/14025 6s 431us/step - accuracy: 0.5107 - loss: 0.9690 - val_accuracy: 0.5199 - val_loss: 0.9523
 Epoch 44/50
14025/14025 6s 419us/step - accuracy: 0.5103 - loss: 0.9695 - val_accuracy: 0.5211 - val_loss: 0.9554
 Epoch 45/50
14025/14025 6s 418us/step - accuracy: 0.5114 - loss: 0.9685 - val_accuracy: 0.5218 - val_loss: 0.9523
 Epoch 46/50
14025/14025 7s 471us/step - accuracy: 0.5109 - loss: 0.9693 - val_accuracy: 0.5198 - val_loss: 0.9518
 Epoch 47/50
14025/14025 6s 424us/step - accuracy: 0.5109 - loss: 0.9699 - val_accuracy: 0.5205 - val_loss: 0.9534
 Epoch 48/50
14025/14025 6s 420us/step - accuracy: 0.5111 - loss: 0.9685 - val_accuracy: 0.5188 - val_loss: 0.9541
 Epoch 49/50
14025/14025 6s 425us/step - accuracy: 0.5112 - loss: 0.9689 - val_accuracy: 0.5213 - val_loss: 0.9522
 Epoch 50/50
14025/14025 6s 448us/step - accuracy: 0.5120 - loss: 0.9688 - val_accuracy: 0.5179 - val_loss: 0.9532
6011/6011 1s 234us/step - accuracy: 0.5167 - loss: 0.9546
6011/6011 1s 213us/step

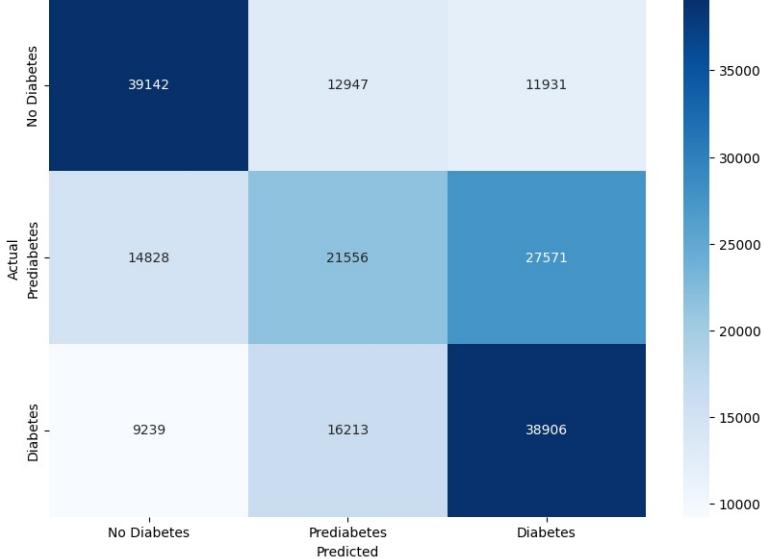
Neural Network Model Evaluation w/ Dropout Layers to Prevent Overfitting
 Accuracy: 0.5178726315498352

precision recall f1-score support

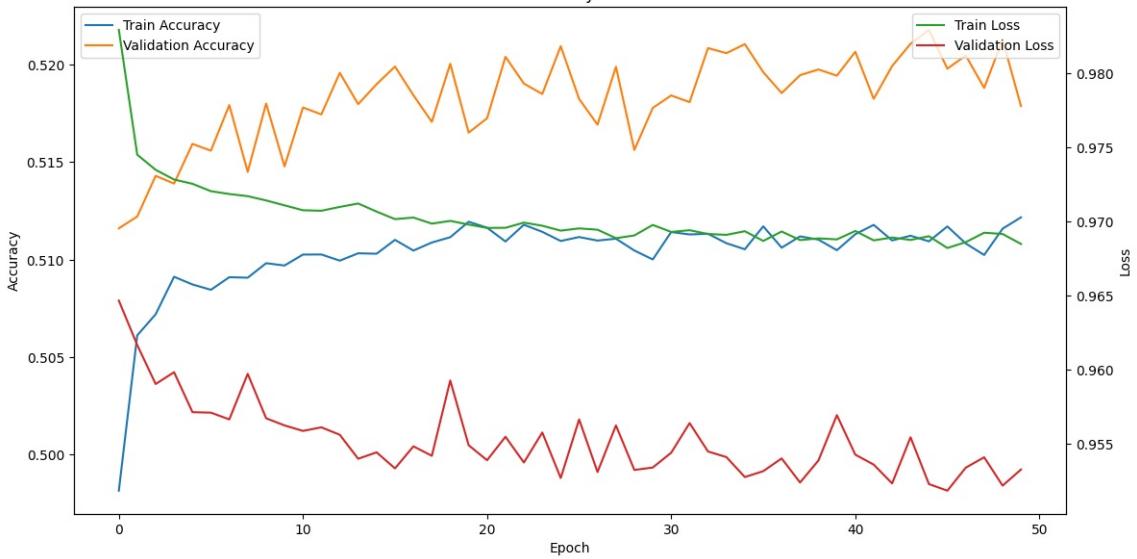
0	0.62	0.61	0.62	64020
1	0.43	0.34	0.38	63955
2	0.50	0.60	0.55	64358

	accuracy	precision	recall	f1-score	support
macro avg	0.51	0.52	0.51	0.52	192333
weighted avg	0.51	0.52	0.51	0.52	192333

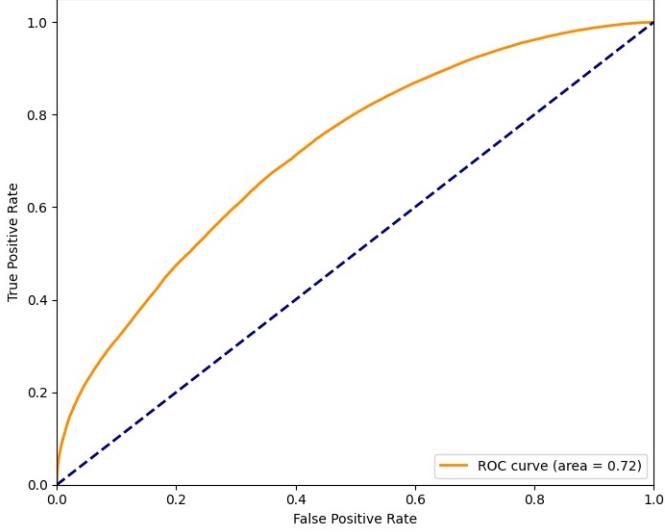
Neural Network Model Evaluation w/ Dropout Layers to Prevent Overfitting



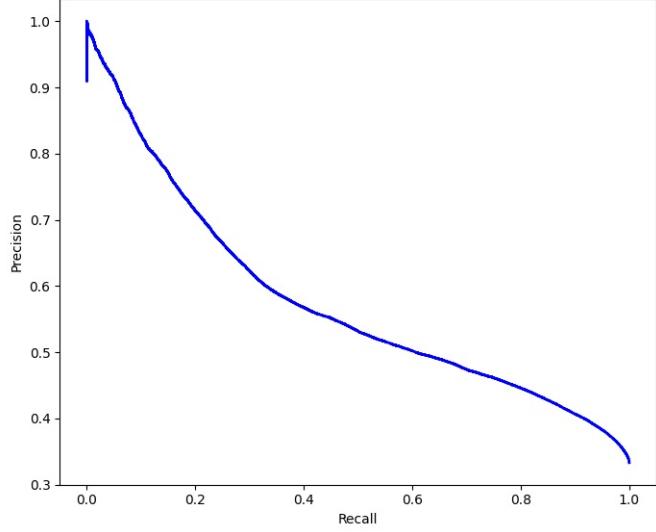
Model Accuracy and Loss



Receiver Operating Characteristic (ROC)



Precision-Recall Curve



Comparative Results Interpretation

Accuracy:

There is a significant decrease in accuracy from 61% to 52%, indicating that the overall performance of the model has worsened.

Class 1 Metrics:

Class 1 has experienced a significant decrease in precision, recall, and F1 score. Precision dropped from 0.59 to 0.43, recall from 0.61 to 0.34, and F1 score from 0.60 to 0.38. This indicates that the model's ability to correctly identify instances of Class 1 has significantly declined.

Class 2 Metrics:

Class 2's recall has slightly increased from 0.58 to 0.60, while the F1 score has slightly decreased from 0.58 to 0.55. The recall improvement indicates a marginally better identification of actual positives for Class 2, but the decrease in F1 score reflects a general decline in model performance across all classes.

Macro Average:

There is a significant decrease in macro average precision, recall, and F1 score, reflecting a general decline in model performance across all classes.

Weighted Average:

Similar to the macro averages, the weighted averages for precision, recall, and F1 score have all significantly decreased. This indicates that the overall balanced performance of the model has degraded, accounting for class distribution.

Adjust Learning Rate Using a Learning Rate Scheduler

A learning rate scheduler adjusts the learning rate during training to help the model converge more efficiently and effectively. This approach prevents overshooting optimal values and can improve model performance by fine-tuning learning rates.

```
In [8]: # Define the learning rate scheduler
def scheduler(epoch, lr):
    if epoch < 10:
        return lr
    else:
        return float(lr * tf.math.exp(-0.1))

lr_scheduler = LearningRateScheduler(scheduler)

# Define the neural network model
model = Sequential()
model.add(Input(shape=(X_train.shape[1],)))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(3, activation='softmax'))

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model with the learning rate scheduler
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test), callbacks=[lr_scheduler])

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_test_classes = np.argmax(y_test, axis=1)

report = classification_report(y_test_classes, y_pred_classes)
title = 'Neural Network Model Evaluation w/ Adjust Learning Rate Using a Learning Rate Scheduler'
display_results(accuracy, report, y_test_classes, y_pred_classes, title)

# Combined Accuracy and Loss Plot
fig, ax1 = plt.subplots(figsize=(12, 6))

ax1.set_xlabel('Epoch')
ax1.set_ylabel('Accuracy')
ax1.plot(history.history['accuracy'], label='Train Accuracy', color='tab:blue')
ax1.plot(history.history['val_accuracy'], label='Validation Accuracy', color='tab:orange')
ax1.tick_params(axis='y')
ax1.legend(loc='upper left')

ax2 = ax1.twinx() # instantiate a second axes that shares the same x-axis
ax2.set_ylabel('Loss')
ax2.plot(history.history['loss'], label='Train Loss', color='tab:green')
ax2.plot(history.history['val_loss'], label='Validation Loss', color='tab:red')
ax2.tick_params(axis='y')
ax2.legend(loc='upper right')

fig.tight_layout() # otherwise the right y-label is slightly clipped
plt.title('Model Accuracy and Loss')
plt.show()

# Calculate ROC curve and ROC AUC
fpr, tpr, _ = roc_curve(y_test.ravel(), y_pred.ravel())
roc_auc = auc(fpr, tpr)

# Calculate Precision-Recall curve
precision, recall, _ = precision_recall_curve(y_test.ravel(), y_pred.ravel())

# Combined ROC and Precision-Recall Plot
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

# ROC Curve
ax1.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
ax1.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
ax1.set_xlim([0.0, 1.0])
ax1.set_ylim([0.0, 1.05])
ax1.set_xlabel('False Positive Rate')
ax1.set_ylabel('True Positive Rate')
ax1.set_title('Receiver Operating Characteristic (ROC)')
ax1.legend(loc='lower right')

# Precision-Recall Curve
ax2.plot(recall, precision, color='blue', lw=2)
ax2.set_xlabel('Recall')
ax2.set_ylabel('Precision')
ax2.set_title('Precision-Recall Curve')

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

Epoch 1/50
14025/14025 6s 396us/step - accuracy: 0.5118 - loss: 0.9623 - val_accuracy: 0.5273 - val_loss: 0.9465 - learning_rate: 0.0010
 Epoch 2/50
14025/14025 7s 470us/step - accuracy: 0.5333 - loss: 0.9423 - val_accuracy: 0.5371 - val_loss: 0.9357 - learning_rate: 0.0010
 Epoch 3/50
14025/14025 6s 390us/step - accuracy: 0.5454 - loss: 0.9295 - val_accuracy: 0.5517 - val_loss: 0.9213 - learning_rate: 0.0010
 Epoch 4/50
14025/14025 5s 389us/step - accuracy: 0.5559 - loss: 0.9210 - val_accuracy: 0.5606 - val_loss: 0.9145 - learning_rate: 0.0010
 Epoch 5/50
14025/14025 5s 389us/step - accuracy: 0.5622 - loss: 0.9120 - val_accuracy: 0.5621 - val_loss: 0.9100 - learning_rate: 0.0010
 Epoch 6/50
14025/14025 5s 387us/step - accuracy: 0.5686 - loss: 0.9055 - val_accuracy: 0.5681 - val_loss: 0.9032 - learning_rate: 0.0010
 Epoch 7/50
14025/14025 5s 386us/step - accuracy: 0.5731 - loss: 0.8979 - val_accuracy: 0.5701 - val_loss: 0.9004 - learning_rate: 0.0010
 Epoch 8/50
14025/14025 6s 392us/step - accuracy: 0.5740 - loss: 0.8957 - val_accuracy: 0.5722 - val_loss: 0.8966 - learning_rate: 0.0010
 Epoch 9/50
14025/14025 5s 384us/step - accuracy: 0.5774 - loss: 0.8932 - val_accuracy: 0.5771 - val_loss: 0.8909 - learning_rate: 0.0010
 Epoch 10/50
14025/14025 5s 387us/step - accuracy: 0.5798 - loss: 0.8881 - val_accuracy: 0.5806 - val_loss: 0.8904 - learning_rate: 0.0010
 Epoch 11/50
14025/14025 5s 387us/step - accuracy: 0.5825 - loss: 0.8851 - val_accuracy: 0.5859 - val_loss: 0.8816 - learning_rate: 9.0484e-04
 Epoch 12/50
14025/14025 5s 387us/step - accuracy: 0.5856 - loss: 0.8819 - val_accuracy: 0.5836 - val_loss: 0.8811 - learning_rate: 8.1873e-04
 Epoch 13/50
14025/14025 5s 387us/step - accuracy: 0.5896 - loss: 0.8770 - val_accuracy: 0.5885 - val_loss: 0.8797 - learning_rate: 7.4082e-04
 Epoch 14/50
14025/14025 5s 388us/step - accuracy: 0.5921 - loss: 0.8708 - val_accuracy: 0.5904 - val_loss: 0.8753 - learning_rate: 6.7032e-04
 Epoch 15/50
14025/14025 5s 389us/step - accuracy: 0.5924 - loss: 0.8716 - val_accuracy: 0.5863 - val_loss: 0.8783 - learning_rate: 6.0653e-04
 Epoch 16/50
14025/14025 5s 385us/step - accuracy: 0.5960 - loss: 0.8678 - val_accuracy: 0.5905 - val_loss: 0.8717 - learning_rate: 5.4881e-04
 Epoch 17/50
14025/14025 5s 386us/step - accuracy: 0.5970 - loss: 0.8661 - val_accuracy: 0.5941 - val_loss: 0.8696 - learning_rate: 4.9659e-04
 Epoch 18/50
14025/14025 5s 385us/step - accuracy: 0.5978 - loss: 0.8642 - val_accuracy: 0.5917 - val_loss: 0.8684 - learning_rate: 4.4933e-04
 Epoch 19/50
14025/14025 5s 385us/step - accuracy: 0.6001 - loss: 0.8619 - val_accuracy: 0.5966 - val_loss: 0.8661 - learning_rate: 4.0657e-04
 Epoch 20/50
14025/14025 5s 388us/step - accuracy: 0.6008 - loss: 0.8599 - val_accuracy: 0.5981 - val_loss: 0.8650 - learning_rate: 3.6788e-04
 Epoch 21/50
14025/14025 6s 401us/step - accuracy: 0.6019 - loss: 0.8577 - val_accuracy: 0.5966 - val_loss: 0.8659 - learning_rate: 3.3287e-04
 Epoch 22/50
14025/14025 5s 391us/step - accuracy: 0.6036 - loss: 0.8569 - val_accuracy: 0.5978 - val_loss: 0.8639 - learning_rate: 3.0119e-04
 Epoch 23/50
14025/14025 6s 425us/step - accuracy: 0.6059 - loss: 0.8545 - val_accuracy: 0.5997 - val_loss: 0.8616 - learning_rate: 2.7253e-04
 Epoch 24/50
14025/14025 6s 434us/step - accuracy: 0.6051 - loss: 0.8550 - val_accuracy: 0.6016 - val_loss: 0.8591 - learning_rate: 2.4660e-04
 Epoch 25/50
14025/14025 6s 399us/step - accuracy: 0.6064 - loss: 0.8520 - val_accuracy: 0.6010 - val_loss: 0.8596 - learning_rate: 2.2313e-04
 Epoch 26/50
14025/14025 5s 384us/step - accuracy: 0.6063 - loss: 0.8532 - val_accuracy: 0.6004 - val_loss: 0.8608 - learning_rate: 2.0190e-04
 Epoch 27/50
14025/14025 6s 404us/step - accuracy: 0.6077 - loss: 0.8517 - val_accuracy: 0.6030 - val_loss: 0.8576 - learning_rate: 1.8268e-04
 Epoch 28/50
14025/14025 6s 395us/step - accuracy: 0.6087 - loss: 0.8497 - val_accuracy: 0.6012 - val_loss: 0.8574 - learning_rate: 1.6530e-04
 Epoch 29/50
14025/14025 5s 390us/step - accuracy: 0.6084 - loss: 0.8496 - val_accuracy: 0.6039 - val_loss: 0.8566 - learning_rate: 1.4957e-04
 Epoch 30/50
14025/14025 5s 387us/step - accuracy: 0.6076 - loss: 0.8506 - val_accuracy: 0.6027 - val_loss: 0.8572 - learning_rate: 1.3534e-04
 Epoch 31/50
14025/14025 6s 396us/step - accuracy: 0.6083 - loss: 0.8486 - val_accuracy: 0.6037 - val_loss: 0.8558 - learning_rate: 1.2246e-04
 Epoch 32/50
14025/14025 5s 390us/step - accuracy: 0.6090 - loss: 0.8495 - val_accuracy: 0.6047 - val_loss: 0.8554 - learning_rate: 1.1080e-04
 Epoch 33/50
14025/14025 5s 390us/step - accuracy: 0.6094 - loss: 0.8472 - val_accuracy: 0.6036 - val_loss: 0.8565 - learning_rate: 1.0026e-04
 Epoch 34/50
14025/14025 5s 387us/step - accuracy: 0.6101 - loss: 0.8448 - val_accuracy: 0.6061 - val_loss: 0.8538 - learning_rate: 9.0718e-05
 Epoch 35/50
14025/14025 6s 393us/step - accuracy: 0.6100 - loss: 0.8466 - val_accuracy: 0.6048 - val_loss: 0.8538 - learning_rate: 8.2085e-05
 Epoch 36/50
14025/14025 5s 388us/step - accuracy: 0.6113 - loss: 0.8455 - val_accuracy: 0.6051 - val_loss: 0.8539 - learning_rate: 7.4274e-05
 Epoch 37/50
14025/14025 6s 393us/step - accuracy: 0.6096 - loss: 0.8469 - val_accuracy: 0.6054 - val_loss: 0.8534 - learning_rate: 6.7206e-05
 Epoch 38/50
14025/14025 5s 387us/step - accuracy: 0.6105 - loss: 0.8463 - val_accuracy: 0.6052 - val_loss: 0.8536 - learning_rate: 6.0810e-05
 Epoch 39/50
14025/14025 5s 388us/step - accuracy: 0.6104 - loss: 0.8462 - val_accuracy: 0.6041 - val_loss: 0.8536 - learning_rate: 5.5023e-05
 Epoch 40/50
14025/14025 5s 391us/step - accuracy: 0.6095 - loss: 0.8473 - val_accuracy: 0.6063 - val_loss: 0.8530 - learning_rate: 4.9787e-05
 Epoch 41/50
14025/14025 5s 390us/step - accuracy: 0.6116 - loss: 0.8450 - val_accuracy: 0.6051 - val_loss: 0.8528 - learning_rate: 4.5049e-05
 Epoch 42/50
14025/14025 6s 406us/step - accuracy: 0.6109 - loss: 0.8453 - val_accuracy: 0.6054 - val_loss: 0.8526 - learning_rate: 4.0762e-05
 Epoch 43/50
14025/14025 5s 387us/step - accuracy: 0.6117 - loss: 0.8444 - val_accuracy: 0.6057 - val_loss: 0.8527 - learning_rate: 3.6883e-05
 Epoch 44/50
14025/14025 5s 382us/step - accuracy: 0.6119 - loss: 0.8443 - val_accuracy: 0.6059 - val_loss: 0.8528 - learning_rate: 3.3373e-05
 Epoch 45/50
14025/14025 6s 437us/step - accuracy: 0.6109 - loss: 0.8454 - val_accuracy: 0.6060 - val_loss: 0.8524 - learning_rate: 3.0197e-05
 Epoch 46/50
14025/14025 6s 396us/step - accuracy: 0.6118 - loss: 0.8447 - val_accuracy: 0.6065 - val_loss: 0.8522 - learning_rate: 2.7324e-05
 Epoch 47/50
14025/14025 6s 408us/step - accuracy: 0.6106 - loss: 0.8446 - val_accuracy: 0.6064 - val_loss: 0.8522 - learning_rate: 2.4724e-05
 Epoch 48/50
14025/14025 5s 386us/step - accuracy: 0.6112 - loss: 0.8447 - val_accuracy: 0.6068 - val_loss: 0.8521 - learning_rate: 2.2371e-05
 Epoch 49/50
14025/14025 5s 385us/step - accuracy: 0.6124 - loss: 0.8437 - val_accuracy: 0.6065 - val_loss: 0.8522 - learning_rate: 2.0242e-05
 Epoch 50/50
14025/14025 6s 392us/step - accuracy: 0.6118 - loss: 0.8436 - val_accuracy: 0.6061 - val_loss: 0.8520 - learning_rate: 1.8316e-05
6011/6011 1s 224us/step - accuracy: 0.6063 - loss: 0.8533
6011/6011 1s 224us/step

Neural Network Model Evaluation w/ Adjust Learning Rate Using a Learning Rate Scheduler

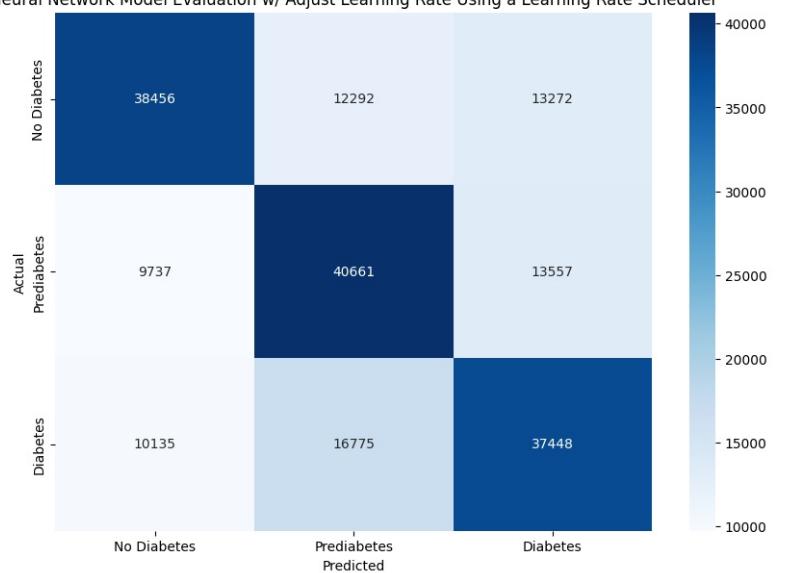
Accuracy: 0.6060582399368286

precision recall f1-score support

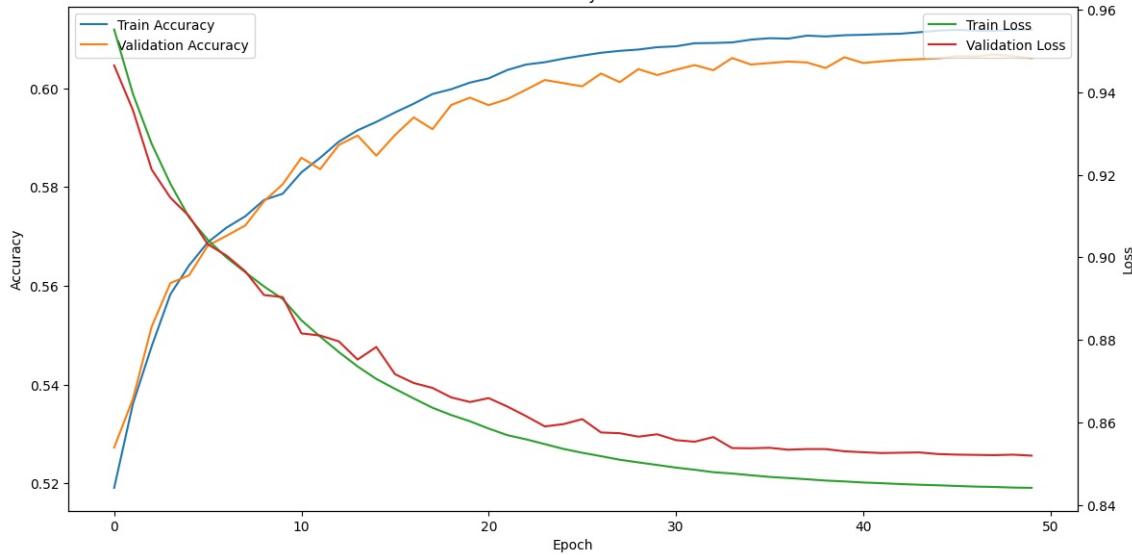
0	0.66	0.60	0.63	64020
1	0.58	0.64	0.61	63955
2	0.58	0.58	0.58	64358

accuracy	0.61	192333
macro avg	0.61	0.61
weighted avg	0.61	0.61

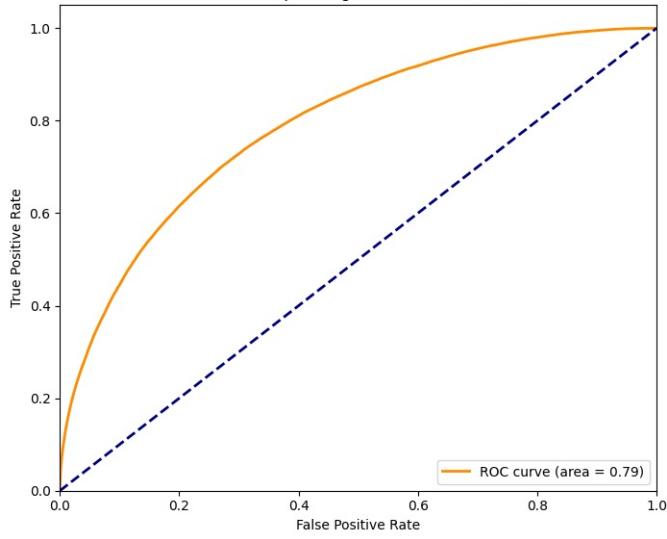
Neural Network Model Evaluation w/ Adjust Learning Rate Using a Learning Rate Scheduler



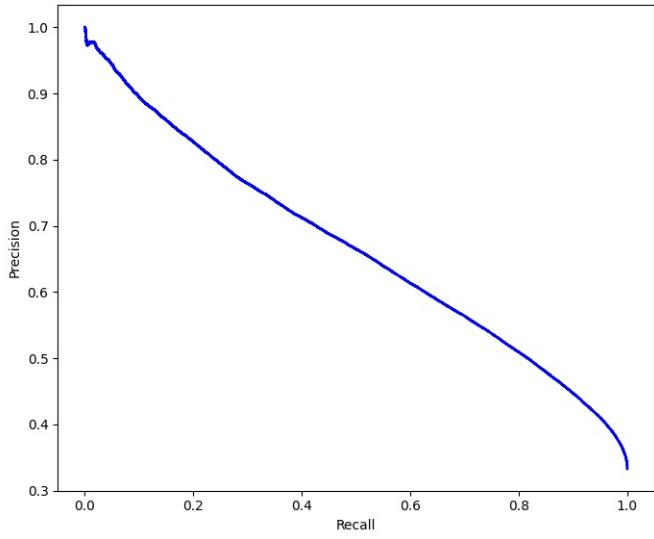
Model Accuracy and Loss



Receiver Operating Characteristic (ROC)



Precision-Recall Curve



Comparative Results Interpretation

Accuracy:

There is a significant increase in accuracy from 52% to 61%, indicating an overall improvement in the model's performance.

Class 1 Metrics:

Class 1 has experienced a significant increase in precision, recall, and F1 score. Precision increased from 0.43 to 0.58, recall from 0.34 to 0.64, and F1 score from 0.38 to 0.61. This indicates that the model's ability to correctly identify instances of Class 1 has improved.

Class 2 Metrics:

Class 2's precision has increased from 0.50 to 0.58, and the F1 score has increased from 0.55 to 0.58. The improvement in precision and F1 score indicates a better balance between precision and recall for Class 2.

Macro Average:

There is a significant increase in macro average precision, recall, and F1 score, reflecting a general improvement in model performance across all classes.

Weighted Average:

Similar to the macro averages, the weighted averages for precision, recall, and F1 score have all significantly increased. This indicates that the overall balanced performance of the model has improved, accounting for class distribution.

Increasing Model Complexity by Adding More Layers and Neurons

Increasing the model complexity by adding more layers and neurons allows the neural network to capture more intricate patterns and relationships within the data. This enhancement can improve the model's ability to learn and generalize from performance.

```
[in [7]: # Define the neural network model with increased complexity
model = Sequential()
model.add(Input(shape=(X_train.shape[1],)))
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(3, activation='softmax'))

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test))

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_test_classes = np.argmax(y_test, axis=1)

report = classification_report(y_test_classes, y_pred_classes)
title = 'Neural Network Model Evaluation w/ More Layers and Neurons'
display_results(accuracy, report, y_test_classes, y_pred_classes, title)

# Combined Accuracy and Loss Plot
fig, ax1 = plt.subplots(figsize=(12, 6))
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Accuracy')
ax1.plot(history.history['accuracy'], label='Train Accuracy', color='tab:blue')
ax1.plot(history.history['val_accuracy'], label='Validation Accuracy', color='tab:orange')
ax1.tick_params(axis='y')
ax1.legend(loc='upper left')

ax2 = ax1.twinx() # instantiate a second axes that shares the same x-axis
ax2.set_ylabel('Loss')
ax2.plot(history.history['loss'], label='Train Loss', color='tab:green')
ax2.plot(history.history['val_loss'], label='Validation Loss', color='tab:red')
ax2.tick_params(axis='y')
ax2.legend(loc='upper right')

fig.tight_layout() # otherwise the right y-label is slightly clipped
plt.title('Model Accuracy and Loss')
plt.show()

# Calculate ROC curve and ROC AUC
fpr, tpr, _ = roc_curve(y_test.ravel(), y_pred.ravel())
roc_auc = auc(fpr, tpr)

# Calculate Precision-Recall curve
precision, recall, _ = precision_recall_curve(y_test.ravel(), y_pred.ravel())

# Combined ROC and Precision-Recall Plot
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

# ROC Curve
ax1.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
ax1.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
ax1.set_xlim([0.0, 1.0])
ax1.set_ylim([0.0, 1.05])
ax1.set_xlabel('False Positive Rate')
ax1.set_ylabel('True Positive Rate')
ax1.set_title('Receiver Operating Characteristic (ROC)')
ax1.legend(loc='lower right')

# Precision-Recall Curve
ax2.plot(recall, precision, color='blue', lw=2)
ax2.set_xlabel('Recall')
ax2.set_ylabel('Precision')
ax2.set_title('Precision-Recall Curve')

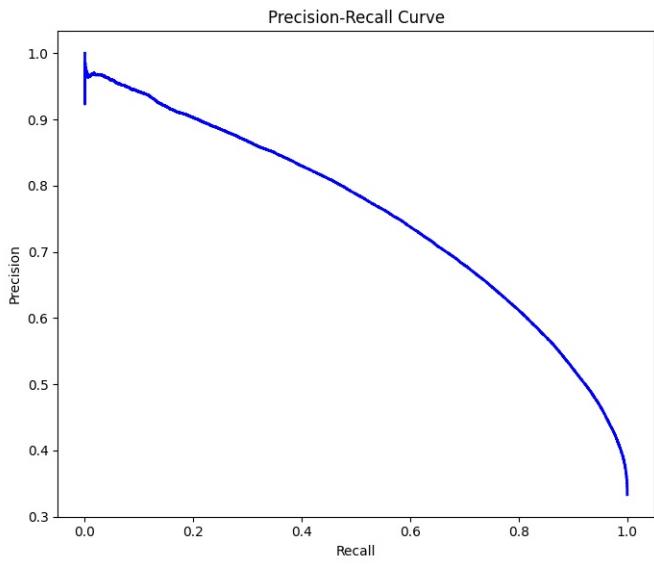
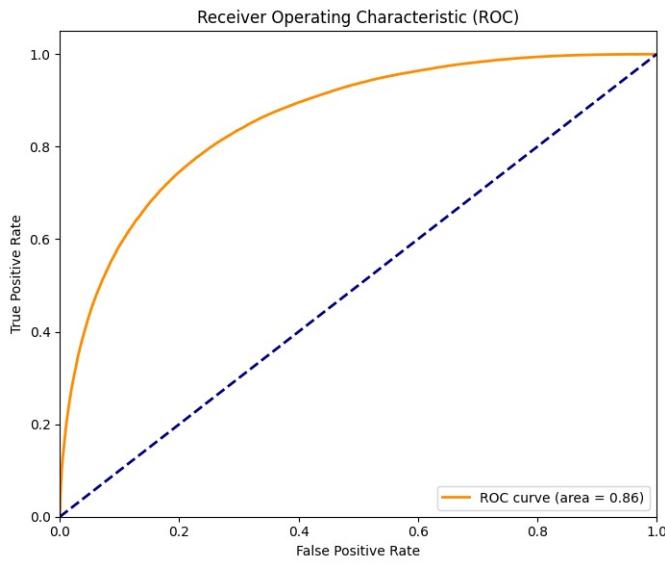
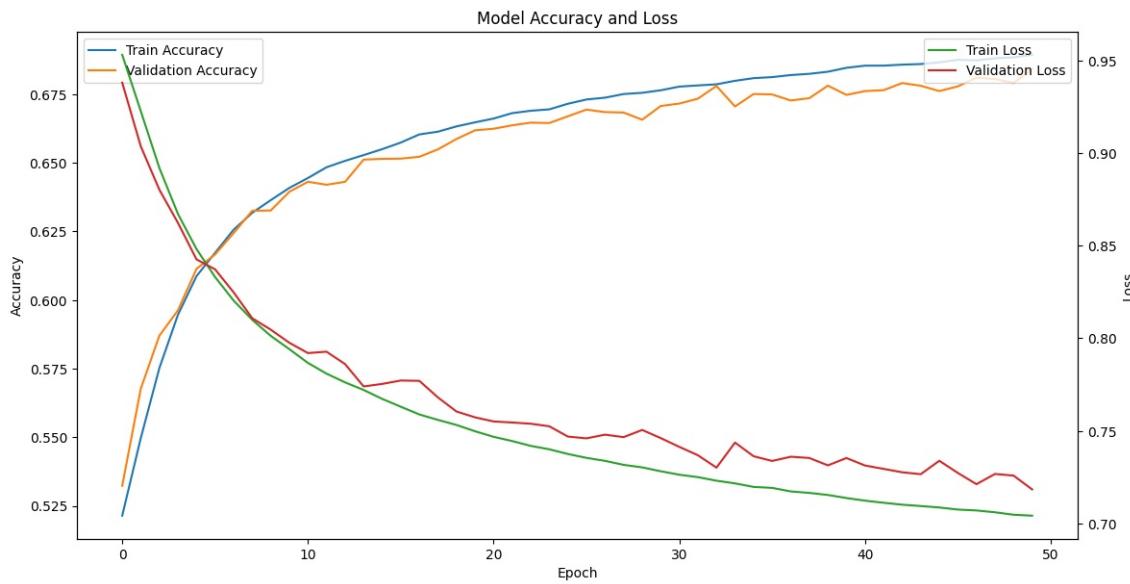
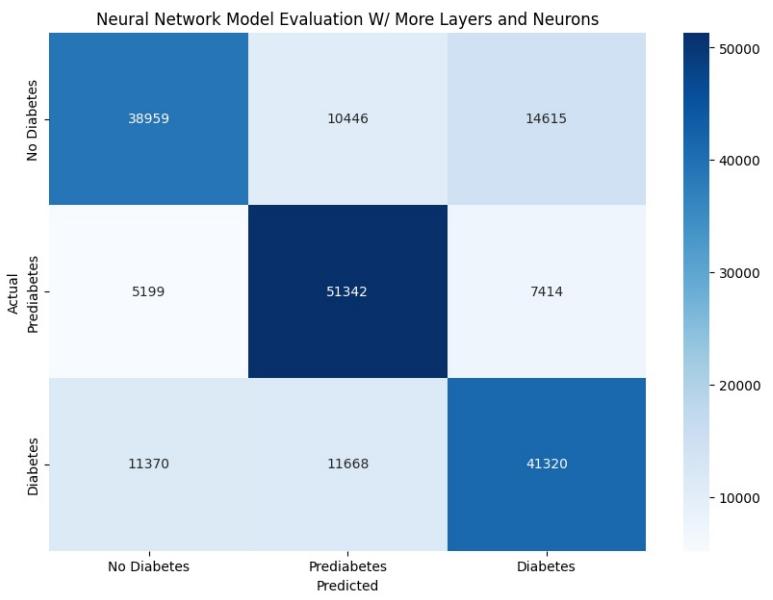
plt.tight_layout()
plt.show()
```

Epoch 1/50
14025/14025 7s 455us/step - accuracy: 0.5137 - loss: 0.9609 - val_accuracy: 0.5323 - val_loss: 0.9380
 Epoch 2/50
14025/14025 6s 447us/step - accuracy: 0.5428 - loss: 0.9302 - val_accuracy: 0.5677 - val_loss: 0.9036
 Epoch 3/50
14025/14025 6s 455us/step - accuracy: 0.5701 - loss: 0.8979 - val_accuracy: 0.5870 - val_loss: 0.8802
 Epoch 4/50
14025/14025 6s 454us/step - accuracy: 0.5914 - loss: 0.8705 - val_accuracy: 0.5962 - val_loss: 0.8623
 Epoch 5/50
14025/14025 6s 446us/step - accuracy: 0.6081 - loss: 0.8506 - val_accuracy: 0.6114 - val_loss: 0.8426
 Epoch 6/50
14025/14025 6s 447us/step - accuracy: 0.6147 - loss: 0.8357 - val_accuracy: 0.6166 - val_loss: 0.8372
 Epoch 7/50
14025/14025 6s 454us/step - accuracy: 0.6255 - loss: 0.8205 - val_accuracy: 0.6244 - val_loss: 0.8248
14025/14025 6s 455us/step - accuracy: 0.6313 - loss: 0.8111 - val_accuracy: 0.6325 - val_loss: 0.8108
 Epoch 9/50
14025/14025 6s 450us/step - accuracy: 0.6352 - loss: 0.8036 - val_accuracy: 0.6326 - val_loss: 0.8046
 Epoch 10/50
14025/14025 6s 453us/step - accuracy: 0.6410 - loss: 0.7947 - val_accuracy: 0.6395 - val_loss: 0.7976
 Epoch 11/50
14025/14025 6s 453us/step - accuracy: 0.6446 - loss: 0.7873 - val_accuracy: 0.6431 - val_loss: 0.7920
 Epoch 12/50
14025/14025 6s 454us/step - accuracy: 0.6477 - loss: 0.7819 - val_accuracy: 0.6420 - val_loss: 0.7928
 Epoch 13/50
14025/14025 7s 492us/step - accuracy: 0.6502 - loss: 0.7768 - val_accuracy: 0.6431 - val_loss: 0.7861
 Epoch 14/50
14025/14025 6s 457us/step - accuracy: 0.6536 - loss: 0.7714 - val_accuracy: 0.6512 - val_loss: 0.7740
 Epoch 15/50
14025/14025 6s 462us/step - accuracy: 0.6549 - loss: 0.7660 - val_accuracy: 0.6515 - val_loss: 0.7754
 Epoch 16/50
14025/14025 7s 530us/step - accuracy: 0.6570 - loss: 0.7632 - val_accuracy: 0.6515 - val_loss: 0.7772
 Epoch 17/50
14025/14025 7s 470us/step - accuracy: 0.6591 - loss: 0.7592 - val_accuracy: 0.6522 - val_loss: 0.7770
 Epoch 18/50
14025/14025 6s 460us/step - accuracy: 0.6613 - loss: 0.7552 - val_accuracy: 0.6549 - val_loss: 0.7681
 Epoch 19/50
14025/14025 6s 454us/step - accuracy: 0.6631 - loss: 0.7528 - val_accuracy: 0.6587 - val_loss: 0.7605
 Epoch 20/50
14025/14025 7s 475us/step - accuracy: 0.6663 - loss: 0.7479 - val_accuracy: 0.6619 - val_loss: 0.7573
 Epoch 21/50
14025/14025 6s 458us/step - accuracy: 0.6668 - loss: 0.7468 - val_accuracy: 0.6624 - val_loss: 0.7551
 Epoch 22/50
14025/14025 7s 471us/step - accuracy: 0.6685 - loss: 0.7439 - val_accuracy: 0.6637 - val_loss: 0.7545
 Epoch 23/50
14025/14025 6s 456us/step - accuracy: 0.6696 - loss: 0.7396 - val_accuracy: 0.6647 - val_loss: 0.7539
 Epoch 24/50
14025/14025 7s 463us/step - accuracy: 0.6696 - loss: 0.7392 - val_accuracy: 0.6645 - val_loss: 0.7525
 Epoch 25/50
14025/14025 6s 458us/step - accuracy: 0.6718 - loss: 0.7357 - val_accuracy: 0.6670 - val_loss: 0.7469
 Epoch 26/50
14025/14025 6s 456us/step - accuracy: 0.6716 - loss: 0.7365 - val_accuracy: 0.6694 - val_loss: 0.7460
 Epoch 27/50
14025/14025 7s 464us/step - accuracy: 0.6738 - loss: 0.7340 - val_accuracy: 0.6685 - val_loss: 0.7480
 Epoch 28/50
14025/14025 6s 458us/step - accuracy: 0.6751 - loss: 0.7304 - val_accuracy: 0.6683 - val_loss: 0.7466
 Epoch 29/50
14025/14025 6s 461us/step - accuracy: 0.6765 - loss: 0.7287 - val_accuracy: 0.6657 - val_loss: 0.7505
 Epoch 30/50
14025/14025 6s 460us/step - accuracy: 0.6772 - loss: 0.7265 - val_accuracy: 0.6707 - val_loss: 0.7461
 Epoch 31/50
14025/14025 7s 463us/step - accuracy: 0.6788 - loss: 0.7238 - val_accuracy: 0.6716 - val_loss: 0.7414
 Epoch 32/50
14025/14025 7s 482us/step - accuracy: 0.6790 - loss: 0.7243 - val_accuracy: 0.6734 - val_loss: 0.7369
 Epoch 33/50
14025/14025 6s 462us/step - accuracy: 0.6787 - loss: 0.7227 - val_accuracy: 0.6780 - val_loss: 0.7302
 Epoch 34/50
14025/14025 7s 466us/step - accuracy: 0.6801 - loss: 0.7218 - val_accuracy: 0.6705 - val_loss: 0.7437
 Epoch 35/50
14025/14025 7s 466us/step - accuracy: 0.6805 - loss: 0.7197 - val_accuracy: 0.6751 - val_loss: 0.7363
 Epoch 36/50
14025/14025 7s 526us/step - accuracy: 0.6820 - loss: 0.7177 - val_accuracy: 0.6749 - val_loss: 0.7338
 Epoch 37/50
14025/14025 7s 470us/step - accuracy: 0.6821 - loss: 0.7170 - val_accuracy: 0.6727 - val_loss: 0.7361
 Epoch 38/50
14025/14025 7s 469us/step - accuracy: 0.6825 - loss: 0.7153 - val_accuracy: 0.6736 - val_loss: 0.7354
 Epoch 39/50
14025/14025 7s 464us/step - accuracy: 0.6842 - loss: 0.7146 - val_accuracy: 0.6781 - val_loss: 0.7314
 Epoch 40/50
14025/14025 7s 464us/step - accuracy: 0.6850 - loss: 0.7115 - val_accuracy: 0.6748 - val_loss: 0.7354
 Epoch 41/50
14025/14025 7s 470us/step - accuracy: 0.6852 - loss: 0.7114 - val_accuracy: 0.6761 - val_loss: 0.7313
 Epoch 42/50
14025/14025 6s 462us/step - accuracy: 0.6858 - loss: 0.7098 - val_accuracy: 0.6765 - val_loss: 0.7295
 Epoch 43/50
14025/14025 6s 460us/step - accuracy: 0.6860 - loss: 0.7093 - val_accuracy: 0.6791 - val_loss: 0.7277
 Epoch 44/50
14025/14025 6s 460us/step - accuracy: 0.6871 - loss: 0.7077 - val_accuracy: 0.6781 - val_loss: 0.7266
 Epoch 45/50
14025/14025 6s 461us/step - accuracy: 0.6859 - loss: 0.7085 - val_accuracy: 0.6762 - val_loss: 0.7339
 Epoch 46/50
14025/14025 7s 471us/step - accuracy: 0.6872 - loss: 0.7073 - val_accuracy: 0.6779 - val_loss: 0.7273
 Epoch 47/50
14025/14025 7s 474us/step - accuracy: 0.6879 - loss: 0.7056 - val_accuracy: 0.6810 - val_loss: 0.7212
 Epoch 48/50
14025/14025 7s 465us/step - accuracy: 0.6889 - loss: 0.7036 - val_accuracy: 0.6806 - val_loss: 0.7268
 Epoch 49/50
14025/14025 6s 460us/step - accuracy: 0.6896 - loss: 0.7020 - val_accuracy: 0.6791 - val_loss: 0.7259
 Epoch 50/50
14025/14025 7s 494us/step - accuracy: 0.6894 - loss: 0.7028 - val_accuracy: 0.6843 - val_loss: 0.7184
6011/6011 1s 237us/step - accuracy: 0.6838 - loss: 0.7202
6011/6011 1s 234us/step

Neural Network Model Evaluation W/ More Layers and Neurons
 Accuracy: 0.684339165687561

	precision	recall	f1-score	support
0	0.70	0.61	0.65	64020
1	0.70	0.80	0.75	63955
2	0.65	0.64	0.65	64358

accuracy	0.68	192333
macro avg	0.68	0.68
weighted avg	0.68	0.68



Comparative Results Interpretation

Accuracy:

There is a significant increase in accuracy from 61% to 68%, indicating an overall improvement in the model's performance.

Class 0 Metrics:

Class 0 has experienced a significant increase in precision from 0.66 to 0.70, and the F1 score has increased from 0.63 to 0.65. This indicates that the model's ability to correctly identify instances of Class 0 has improved.

Class 1 Metrics:

Class 1 has experienced a significant increase in precision, recall, and F1 score. Precision increased from 0.58 to 0.70, recall from 0.64 to 0.80, and F1 score from 0.61 to 0.75. This indicates that the model's ability to correctly identify and accurately predict Class 1 instances has improved.

Class 2 Metrics:

Class 2 has experienced a significant increase in precision from 0.58 to 0.65 and F1 score from 0.58 to 0.65. This indicates a better balance between precision and recall for Class 2.

Macro Average:

There is a significant increase in macro average precision, recall, and F1 score, reflecting a general improvement in model performance across all classes.

Weighted Average:

Similar to the macro averages, the weighted averages for precision, recall, and F1 score have all significantly increased. This indicates that the overall balanced performance of the model has improved, accounting for class distribution.

Neural Network Model with Combined Feature Selection and Advanced Training Techniques

Combined all advanced techniques, including feature selection with RandomForest, dropout layers to prevent overfitting, learning rate scheduler, and adding more neurons and layers to create a comprehensive neural network model. The goal is to enhance the model's performance and generalization by leveraging their individual strengths.

```
In [6]: # Define the learning rate scheduler
def scheduler(epoch, lr):
    if epoch < 10:
        return lr
    else:
        return float(lr * tf.math.exp(-0.1))

lr_scheduler = LearningRateScheduler(scheduler)

# Define early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# Define the neural network model with increased complexity and dropout layers
model = Sequential()
model.add(Input(shape=(X_train.shape[1],)))
model.add(Dense(128, activation='relu')) # Adjusted dropout rate
model.add(Dropout(0.4)) # Adjusted dropout rate
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.4)) # Adjusted dropout rate
model.add(Dense(32, activation='relu'))
model.add(Dense(3, activation='softmax'))

# Compile the model with a different optimizer
model.compile(loss='categorical_crossentropy', optimizer=tf.keras.optimizers.Adam(learning_rate=0.001), metrics=['accuracy'])

# Train the model with the learning rate scheduler and early stopping
history = model.fit(X_train, y_train, epochs=200, batch_size=64, validation_data=(X_test, y_test),
                     callbacks=[lr_scheduler, early_stopping])

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_test_classes = np.argmax(y_test, axis=1)

report = classification_report(y_test_classes, y_pred_classes)
title = 'Neural Network Model Evaluation With Combined Feature Selection and Advanced Training Techniques'
display_results(accuracy, report, y_test_classes, y_pred_classes, title)

# Combined Accuracy and Loss Plot
fig, ax1 = plt.subplots(figsize=(12, 6))

ax1.set_xlabel('Epoch')
ax1.set_ylabel('Accuracy')
ax1.plot(history.history['accuracy'], label='Train Accuracy', color='tab:blue')
ax1.plot(history.history['val_accuracy'], label='Validation Accuracy', color='tab:orange')
ax1.tick_params(axis='y')
ax1.legend(loc='upper left')

ax2 = ax1.twinx() # instantiate a second axes that shares the same x-axis
ax2.set_ylabel('Loss')
ax2.plot(history.history['loss'], label='Train Loss', color='tab:green')
ax2.plot(history.history['val_loss'], label='Validation Loss', color='tab:red')
ax2.tick_params(axis='y')
ax2.legend(loc='upper right')

fig.tight_layout() # otherwise the right y-label is slightly clipped
plt.title('Model Accuracy and Loss')
plt.show()

# Calculate ROC curve and ROC AUC
fpr, tpr, _ = roc_curve(y_test.ravel(), y_pred.ravel())
roc_auc = auc(fpr, tpr)

# Calculate Precision-Recall curve
precision, recall, _ = precision_recall_curve(y_test.ravel(), y_pred.ravel())

# Combined ROC and Precision-Recall Plot
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

# ROC Curve
ax1.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
ax1.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
ax1.set_xlim([0.0, 1.0])
ax1.set_ylim([0.0, 1.05])
ax1.set_xlabel('False Positive Rate')
ax1.set_ylabel('True Positive Rate')
ax1.set_title('Receiver Operating Characteristic (ROC)')
ax1.legend(loc='lower right')

# Precision-Recall Curve
ax2.plot(recall, precision, color='blue', lw=2)
ax2.set_xlabel('Recall')
ax2.set_ylabel('Precision')
ax2.set_title('Precision-Recall Curve')

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

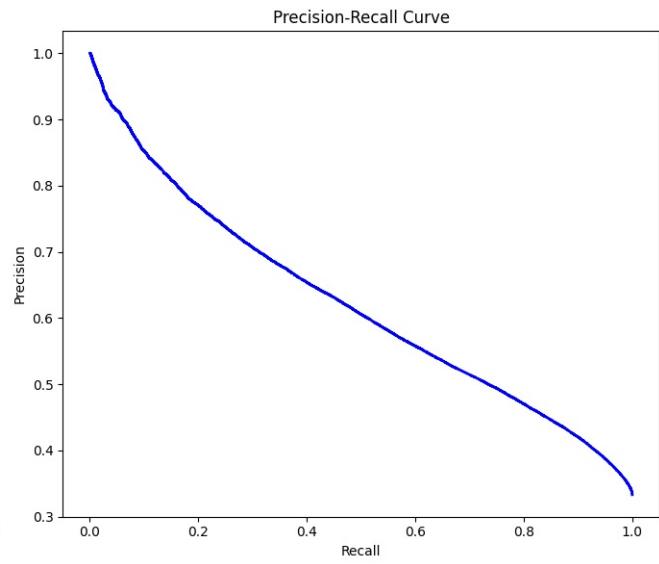
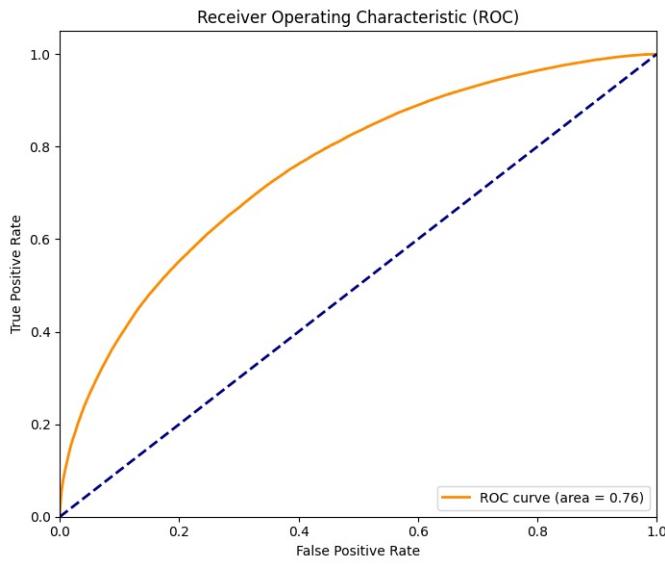
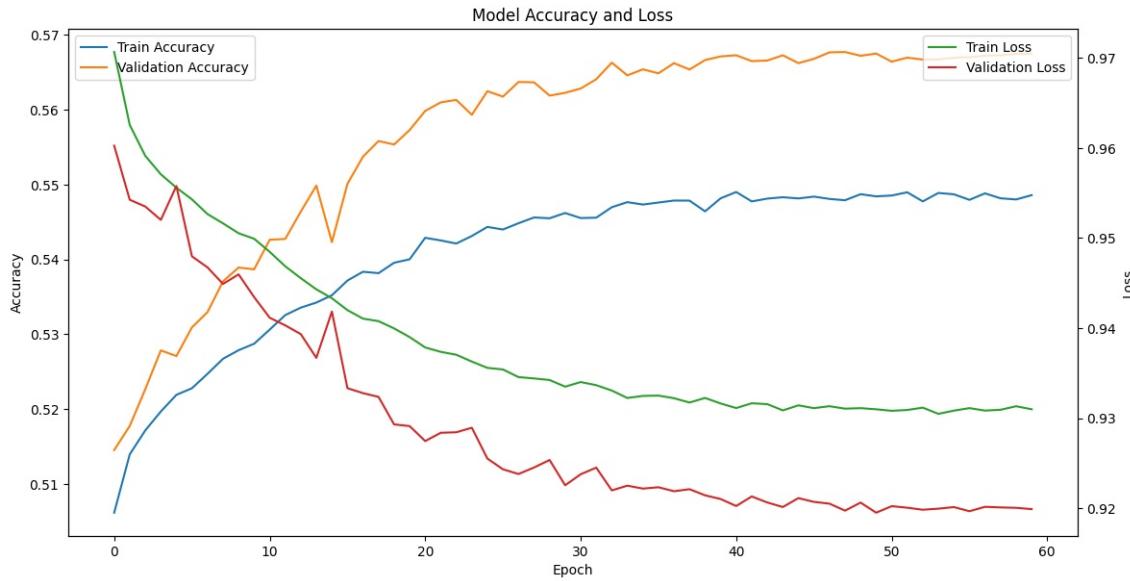
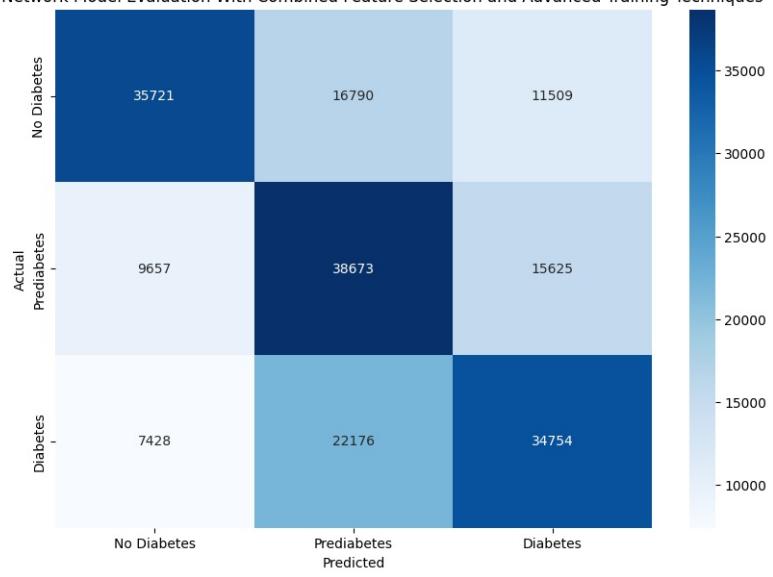
Epoch 1/200
7013/7013 5s 589us/step - accuracy: 0.4992 - loss: 0.9790 - val_accuracy: 0.5146 - val_loss: 0.9603 - learning_rate: 0.0010
 Epoch 2/200
7013/7013 4s 615us/step - accuracy: 0.5124 - loss: 0.9636 - val_accuracy: 0.5178 - val_loss: 0.9543 - learning_rate: 0.0010
 Epoch 3/200
7013/7013 4s 549us/step - accuracy: 0.5152 - loss: 0.9607 - val_accuracy: 0.5227 - val_loss: 0.9535 - learning_rate: 0.0010
 Epoch 4/200
7013/7013 4s 562us/step - accuracy: 0.5196 - loss: 0.9571 - val_accuracy: 0.5279 - val_loss: 0.9520 - learning_rate: 0.0010
 Epoch 5/200
7013/7013 4s 566us/step - accuracy: 0.5224 - loss: 0.9547 - val_accuracy: 0.5271 - val_loss: 0.9558 - learning_rate: 0.0010
 Epoch 6/200
7013/7013 5s 659us/step - accuracy: 0.5219 - loss: 0.9546 - val_accuracy: 0.5309 - val_loss: 0.9480 - learning_rate: 0.0010
 Epoch 7/200
7013/7013 4s 573us/step - accuracy: 0.5253 - loss: 0.9533 - val_accuracy: 0.5330 - val_loss: 0.9467 - learning_rate: 0.0010
7013/7013 4s 585us/step - accuracy: 0.5259 - loss: 0.9517 - val_accuracy: 0.5371 - val_loss: 0.9449 - learning_rate: 0.0010
 Epoch 9/200
7013/7013 4s 557us/step - accuracy: 0.5280 - loss: 0.9507 - val_accuracy: 0.5389 - val_loss: 0.9460 - learning_rate: 0.0010
 Epoch 10/200
7013/7013 5s 721us/step - accuracy: 0.5289 - loss: 0.9494 - val_accuracy: 0.5387 - val_loss: 0.9434 - learning_rate: 0.0010
 Epoch 11/200
7013/7013 4s 587us/step - accuracy: 0.5303 - loss: 0.9479 - val_accuracy: 0.5426 - val_loss: 0.9412 - learning_rate: 0.0484e-04
 Epoch 12/200
7013/7013 4s 551us/step - accuracy: 0.5323 - loss: 0.9461 - val_accuracy: 0.5427 - val_loss: 0.9403 - learning_rate: 8.1873e-04
 Epoch 13/200
7013/7013 4s 556us/step - accuracy: 0.5328 - loss: 0.9462 - val_accuracy: 0.5464 - val_loss: 0.9393 - learning_rate: 7.4082e-04
 Epoch 14/200
7013/7013 4s 544us/step - accuracy: 0.5349 - loss: 0.9430 - val_accuracy: 0.5499 - val_loss: 0.9367 - learning_rate: 6.7032e-04
 Epoch 15/200
7013/7013 4s 562us/step - accuracy: 0.5345 - loss: 0.9442 - val_accuracy: 0.5423 - val_loss: 0.9418 - learning_rate: 6.0653e-04
 Epoch 16/200
7013/7013 4s 572us/step - accuracy: 0.5375 - loss: 0.9416 - val_accuracy: 0.5501 - val_loss: 0.9333 - learning_rate: 5.4881e-04
 Epoch 17/200
7013/7013 4s 556us/step - accuracy: 0.5397 - loss: 0.9411 - val_accuracy: 0.5537 - val_loss: 0.9328 - learning_rate: 4.9659e-04
 Epoch 18/200
7013/7013 4s 568us/step - accuracy: 0.5375 - loss: 0.9416 - val_accuracy: 0.5558 - val_loss: 0.9324 - learning_rate: 4.4933e-04
 Epoch 19/200
7013/7013 4s 562us/step - accuracy: 0.5397 - loss: 0.9404 - val_accuracy: 0.5554 - val_loss: 0.9293 - learning_rate: 4.0657e-04
 Epoch 20/200
7013/7013 4s 564us/step - accuracy: 0.5410 - loss: 0.9380 - val_accuracy: 0.5573 - val_loss: 0.9291 - learning_rate: 3.6788e-04
 Epoch 21/200
7013/7013 4s 599us/step - accuracy: 0.5429 - loss: 0.9375 - val_accuracy: 0.5598 - val_loss: 0.9275 - learning_rate: 3.3287e-04
 Epoch 22/200
7013/7013 4s 580us/step - accuracy: 0.5423 - loss: 0.9372 - val_accuracy: 0.5610 - val_loss: 0.9284 - learning_rate: 3.0119e-04
 Epoch 23/200
7013/7013 4s 570us/step - accuracy: 0.5426 - loss: 0.9366 - val_accuracy: 0.5613 - val_loss: 0.9284 - learning_rate: 2.7253e-04
 Epoch 24/200
7013/7013 4s 569us/step - accuracy: 0.5425 - loss: 0.9361 - val_accuracy: 0.5593 - val_loss: 0.9289 - learning_rate: 2.4660e-04
 Epoch 25/200
7013/7013 4s 576us/step - accuracy: 0.5441 - loss: 0.9363 - val_accuracy: 0.5625 - val_loss: 0.9255 - learning_rate: 2.2313e-04
 Epoch 26/200
7013/7013 4s 569us/step - accuracy: 0.5433 - loss: 0.9349 - val_accuracy: 0.5617 - val_loss: 0.9243 - learning_rate: 2.0190e-04
 Epoch 27/200
7013/7013 4s 576us/step - accuracy: 0.5438 - loss: 0.9356 - val_accuracy: 0.5637 - val_loss: 0.9238 - learning_rate: 1.8268e-04
 Epoch 28/200
7013/7013 4s 574us/step - accuracy: 0.5470 - loss: 0.9336 - val_accuracy: 0.5637 - val_loss: 0.9245 - learning_rate: 1.6530e-04
 Epoch 29/200
7013/7013 4s 570us/step - accuracy: 0.5453 - loss: 0.9349 - val_accuracy: 0.5619 - val_loss: 0.9254 - learning_rate: 1.4957e-04
 Epoch 30/200
7013/7013 4s 579us/step - accuracy: 0.5472 - loss: 0.9333 - val_accuracy: 0.5623 - val_loss: 0.9225 - learning_rate: 1.3534e-04
 Epoch 31/200
7013/7013 4s 585us/step - accuracy: 0.5456 - loss: 0.9339 - val_accuracy: 0.5628 - val_loss: 0.9238 - learning_rate: 1.2246e-04
 Epoch 32/200
7013/7013 4s 578us/step - accuracy: 0.5456 - loss: 0.9329 - val_accuracy: 0.5641 - val_loss: 0.9245 - learning_rate: 1.1080e-04
 Epoch 33/200
7013/7013 4s 590us/step - accuracy: 0.5480 - loss: 0.9321 - val_accuracy: 0.5663 - val_loss: 0.9220 - learning_rate: 1.0026e-04
 Epoch 34/200
7013/7013 4s 578us/step - accuracy: 0.5489 - loss: 0.9321 - val_accuracy: 0.5646 - val_loss: 0.9225 - learning_rate: 9.0718e-05
 Epoch 35/200
7013/7013 4s 582us/step - accuracy: 0.5461 - loss: 0.9339 - val_accuracy: 0.5654 - val_loss: 0.9222 - learning_rate: 8.2085e-05
 Epoch 36/200
7013/7013 4s 614us/step - accuracy: 0.5462 - loss: 0.9326 - val_accuracy: 0.5649 - val_loss: 0.9223 - learning_rate: 7.4274e-05
 Epoch 37/200
7013/7013 4s 585us/step - accuracy: 0.5471 - loss: 0.9328 - val_accuracy: 0.5662 - val_loss: 0.9219 - learning_rate: 6.7206e-05
 Epoch 38/200
7013/7013 4s 566us/step - accuracy: 0.5491 - loss: 0.9314 - val_accuracy: 0.5654 - val_loss: 0.9221 - learning_rate: 6.0810e-05
 Epoch 39/200
7013/7013 4s 627us/step - accuracy: 0.5469 - loss: 0.9320 - val_accuracy: 0.5666 - val_loss: 0.9214 - learning_rate: 5.5023e-05
 Epoch 40/200
7013/7013 5s 636us/step - accuracy: 0.5493 - loss: 0.9304 - val_accuracy: 0.5671 - val_loss: 0.9210 - learning_rate: 4.9787e-05
 Epoch 41/200
7013/7013 4s 584us/step - accuracy: 0.5490 - loss: 0.9322 - val_accuracy: 0.5673 - val_loss: 0.9202 - learning_rate: 4.5049e-05
 Epoch 42/200
7013/7013 4s 591us/step - accuracy: 0.5485 - loss: 0.9309 - val_accuracy: 0.5665 - val_loss: 0.9213 - learning_rate: 4.0762e-05
 Epoch 43/200
7013/7013 4s 564us/step - accuracy: 0.5485 - loss: 0.9312 - val_accuracy: 0.5666 - val_loss: 0.9206 - learning_rate: 3.6883e-05
 Epoch 44/200
7013/7013 4s 561us/step - accuracy: 0.5492 - loss: 0.9304 - val_accuracy: 0.5673 - val_loss: 0.9201 - learning_rate: 3.3373e-05
 Epoch 45/200
7013/7013 4s 561us/step - accuracy: 0.5482 - loss: 0.9314 - val_accuracy: 0.5662 - val_loss: 0.9211 - learning_rate: 3.0197e-05
 Epoch 46/200
7013/7013 4s 561us/step - accuracy: 0.5480 - loss: 0.9312 - val_accuracy: 0.5668 - val_loss: 0.9207 - learning_rate: 2.7324e-05
 Epoch 47/200
7013/7013 4s 561us/step - accuracy: 0.5480 - loss: 0.9314 - val_accuracy: 0.5677 - val_loss: 0.9205 - learning_rate: 2.4724e-05
 Epoch 48/200
7013/7013 4s 562us/step - accuracy: 0.5470 - loss: 0.9317 - val_accuracy: 0.5677 - val_loss: 0.9197 - learning_rate: 2.2371e-05
 Epoch 49/200
7013/7013 4s 572us/step - accuracy: 0.5495 - loss: 0.9307 - val_accuracy: 0.5672 - val_loss: 0.9206 - learning_rate: 2.0242e-05
 Epoch 50/200
7013/7013 4s 573us/step - accuracy: 0.5488 - loss: 0.9308 - val_accuracy: 0.5675 - val_loss: 0.9195 - learning_rate: 1.8316e-05
 Epoch 51/200
7013/7013 4s 561us/step - accuracy: 0.5493 - loss: 0.9304 - val_accuracy: 0.5664 - val_loss: 0.9202 - learning_rate: 1.6573e-05
 Epoch 52/200
7013/7013 4s 557us/step - accuracy: 0.5483 - loss: 0.9323 - val_accuracy: 0.5669 - val_loss: 0.9201 - learning_rate: 1.4996e-05
 Epoch 53/200
7013/7013 4s 557us/step - accuracy: 0.5473 - loss: 0.9315 - val_accuracy: 0.5667 - val_loss: 0.9198 - learning_rate: 1.3569e-05
 Epoch 54/200
7013/7013 4s 559us/step - accuracy: 0.5486 - loss: 0.9301 - val_accuracy: 0.5667 - val_loss: 0.9199 - learning_rate: 1.2277e-05
 Epoch 55/200
7013/7013 4s 561us/step - accuracy: 0.5503 - loss: 0.9293 - val_accuracy: 0.5669 - val_loss: 0.9201 - learning_rate: 1.1109e-05
 Epoch 56/200
7013/7013 4s 558us/step - accuracy: 0.5486 - loss: 0.9321 - val_accuracy: 0.5670 - val_loss: 0.9197 - learning_rate: 1.0052e-05
 Epoch 57/200
7013/7013 4s 557us/step - accuracy: 0.5496 - loss: 0.9302 - val_accuracy: 0.5672 - val_loss: 0.9201 - learning_rate: 9.0953e-06
 Epoch 58/200
7013/7013 4s 558us/step - accuracy: 0.5485 - loss: 0.9301 - val_accuracy: 0.5673 - val_loss: 0.9201 - learning_rate: 8.2298e-06
 Epoch 59/200
7013/7013 4s 555us/step - accuracy: 0.5482 - loss: 0.9307 - val_accuracy: 0.5675 - val_loss: 0.9200 - learning_rate: 7.4466e-06
 Epoch 60/200
7013/7013 4s 557us/step - accuracy: 0.5484 - loss: 0.9313 - val_accuracy: 0.5675 - val_loss: 0.9199 - learning_rate: 6.7379e-06
 6011/6011 1s 218us/step - accuracy: 0.5675 - loss: 0.9201
 6011/6011 1s 223us/step - accuracy: 0.5674949288368225

Neural Network Model Evaluation With Combined Feature Selection and Advanced Training Techniques
 Accuracy: 0.5674949288368225

	precision	recall	f1-score	support
accuracy	0.68	0.56	0.61	64020
1	0.50	0.60	0.55	63955
2	0.56	0.54	0.55	64358

	accuracy	macro avg	weighted avg
precision	0.57	0.58	0.58
recall	0.57	0.57	0.57
f1-score	0.57	0.57	0.57
support	192333	192333	192333

Neural Network Model Evaluation With Combined Feature Selection and Advanced Training Techniques



Comparative Results Interpretation

Accuracy:

There is a significant decrease in accuracy from 68% to 57%, indicating an overall decline in the model's performance.

Class 0 Metrics:

Class 0 has experienced a decrease in precision from 0.70 to 0.68, recall from 0.61 to 0.56, and F1 score from 0.65 to 0.61. This indicates a decline in the model's ability to correctly identify and accurately classify instances of Class 0.

Class 1 Metrics:

Class 1 has experienced a significant decrease in precision, recall, and F1 score. Precision decreased from 0.70 to 0.50, recall from 0.80 to 0.60, and F1 score from 0.75 to 0.55. This indicates a substantial decline in the model's ability to correctly identify and accurately classify instances of Class 1.

Class 2 Metrics:

Class 2 has experienced a decrease in precision from 0.65 to 0.56, recall from 0.64 to 0.54, and F1 score from 0.65 to 0.55. This indicates a decline in the model's ability to correctly identify and accurately classify instances of Class 2.

Macro Average:

There is a significant decrease in macro average precision, recall, and F1 score, reflecting a general decline in model performance across all classes.

Weighted Average:

Similar to the macro averages, the weighted averages for precision, recall, and F1 score have all significantly decreased. This indicates that the overall balanced performance of the model has declined, accounting for class distribution.

Neural Network Conclusion

The original model demonstrated balanced recall values across all classes, averaging around 60%. However, after applying combined feature selection and advanced training techniques, the recall values decreased for all classes. This suggests the model's ability to correctly identify positive instances. The overall low recall can be attributed to the linear relationship demonstrated in the EDA.

XGBoost

```
In [73]: # Getting a clean set and checking for Missing Values and Place a Random Seed
np.random.seed(42)

# Test train split baseline model
X = data.drop(columns=['Diabetes_012'])
y = data['Diabetes_012']

X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size=0.2, random_state=42, stratify=y_temp)

# Show the class distribution pre-imbalance address
print("Original class distribution:\n", y_train.value_counts())

# Function to preprocess the data
def preprocess_data(X_train, X_val, X_test, y_train):
    # Impute missing values
    imputer = SimpleImputer(strategy='median')
    X_train_imputed = imputer.fit_transform(X_train)
    X_val_imputed = imputer.transform(X_val)
    X_test_imputed = imputer.transform(X_test)

    # Apply SMOTE to the training data only
    smote = SMOTE(random_state=42)
    X_train_smote, y_train_smote = smote.fit_resample(X_train_imputed, y_train)

    # Determine the desired sampling strategy for undersampling
    class_counts = y_train_smote.value_counts()
    desired_samples_per_class = class_counts.min()
    undersample_strategy = {label: desired_samples_per_class for label in class_counts.index}

    # Apply undersampling to the data after SMOTE
    undersample = RandomUnderSampler(sampling_strategy=undersample_strategy, random_state=42)
    X_train_res, y_train_res = undersample.fit_resample(X_train_smote, y_train_smote)

    # Apply PCA
    pca = PCA(n_components=15)
    X_train_res_pca = pca.fit_transform(X_train_res)
    X_val_pca = pca.transform(X_val_imputed)
    X_test_pca = pca.transform(X_test_imputed)

    return X_train_res_pca, X_val_pca, X_test_pca, y_train_res, y_val, y_test

# Preprocess the data
X_train_res_pca, X_val_pca, X_test_pca, y_train_res, y_val, y_test = preprocess_data(X_train, X_val, X_test, y_train)

# Baseline Model
xgboost = XGBClassifier(use_label_encoder=True, eval_metric='logloss')
xgboost.fit(X_train_res_pca, y_train_res)

# Prediction
y_val_pred = xgboost.predict(X_val_pca)

# Evaluate the model
precision = precision_score(y_val, y_val_pred, average='weighted')
recall = recall_score(y_val, y_val_pred, average='weighted')
f1 = f1_score(y_val, y_val_pred, average='weighted')
baseline_report = classification_report(y_val, y_val_pred)

print(baseline_report)
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1 Score: {f1}')

# Generate confusion matrix
cm = confusion_matrix(y_val, y_val_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=xgboost.classes_)
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.show()

# Plot ROC curve
plot_ROC_curve(xgboost, X_train_res_pca, y_train_res, X_val_pca, y_val)

# Hyperparameter Tuning for XGBoost Using Optuna
def objective(trial):
    param = {
        'n_estimators': trial.suggest_int('n_estimators', 50, 300),
        'max_depth': trial.suggest_int('max_depth', 3, 10),
        'learning_rate': trial.suggest_float('learning_rate', 0.01, 0.2, log=True),
        'subsample': trial.suggest_float('subsample', 0.6, 1.0),
        'colsample_bytree': trial.suggest_float('colsample_bytree', 0.6, 1.0),
        'gamma': trial.suggest_float('gamma', 0.1, 0.4, log=True),
        'min_child_weight': trial.suggest_int('min_child_weight', 1, 5)
    }

    xgb_model = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', **param)

    scoring = {
        'accuracy': 'accuracy',
        'precision': 'precision_weighted',
        'recall': 'recall_weighted',
        'f1': 'f1_weighted'
    }

    scores = cross_validate(xgb_model, X_train_res_pca, y_train_res, cv=3, scoring=scoring)

    trial.set_user_attr('accuracy', scores['test_accuracy'].mean())
    trial.set_user_attr('precision', scores['test_precision'].mean())
    trial.set_user_attr('recall', scores['test_recall'].mean())
    trial.set_user_attr('f1', scores['test_f1'].mean())

    return scores['test_f1'].mean()

db_path = "xgboost_study.db"
db_url = f"sqlite:///{db_path}"
pruner = optuna.pruners.MedianPruner()
study = optuna.create_study(direction='maximize', storage=db_url, study_name="xgb_study", pruner=pruner, load_if_exists=True)

n_trials = 50
study.optimize(objective, n_trials=n_trials)

print("Best trial:")
best_trial = study.best_trial
print(f" Value (F1 Score): {best_trial.value}")
print(" Params:")
for key, value in best_trial.params.items():
    print(f" {key}: {value}")

print(" Accuracy: {best_trial.user_attrs['accuracy']}")
print(" Precision: {best_trial.user_attrs['precision']}")
print(" Recall: {best_trial.user_attrs['recall']}")
print(" F1 Score: {best_trial.user_attrs['f1']}")

# Initialize the final XGBoost model with the best parameters
final_xgb_model = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', **best_trial.params)
final_xgb_model.fit(X_train_res_pca, y_train_res)
```

```

# Make predictions on the validation set
y_val_pred = final_xgb_model.predict(X_val_pca)

# Evaluate the model using multiple metrics
precision = precision_score(y_val, y_val_pred, average='weighted')
recall = recall_score(y_val, y_val_pred, average='weighted')
f1 = f1_score(y_val, y_val_pred, average='weighted')
accuracy = final_xgb_model.score(X_val_pca, y_val)

print(classification_report(y_val, y_val_pred))
print("Accuracy: [accuracy]")
print("Precision: [precision]")
print("Recall: [recall]")
print("F1 Score: [f1]")

# Generate confusion matrix
cm = confusion_matrix(y_val, y_val_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=final_xgb_model.classes_)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.show()

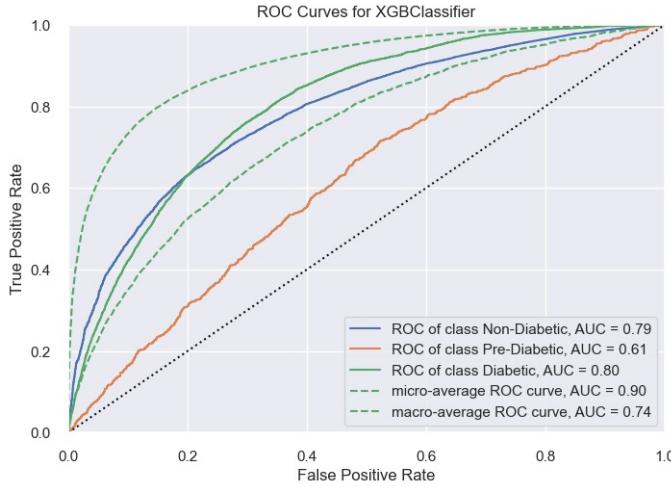
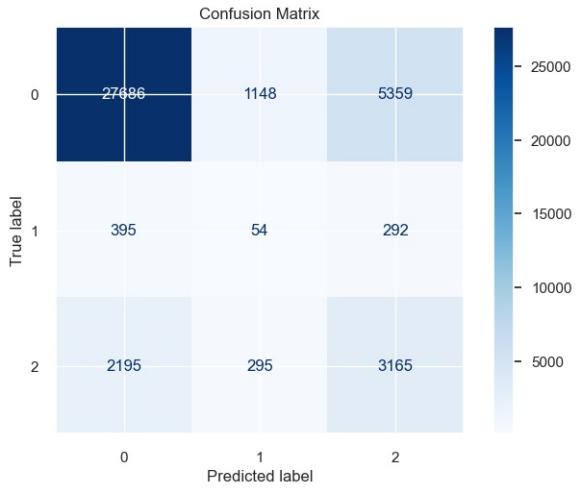
# Plot ROC curve
plot_ROC_curve(final_xgb_model, X_train_res_pca, y_train_res, X_val_pca, y_val)

Original class distribution:
Diabetes_012
0.0 136769
2.0 22622
1.0 2964
Name: count, dtype: int64
   precision    recall  f1-score   support
0.0      0.91     0.81     0.86   34193
1.0      0.04     0.07     0.05     741
2.0      0.36     0.56     0.44   5655

accuracy           0.76    40589
macro avg       0.44     0.48     0.45    40589
weighted avg    0.82     0.76     0.79    40589

Precision: 0.8210309565215307
Recall: 0.761413190765971
F1 Score: 0.785374085935024

```



[I 2024-06-23 17:16:10.009] A new study created in RDB with name: xgb_study

[I 2024-06-23 17:17:23.053] Trial 0 finished with value: 0.7637459448936751 and parameters: {'n_estimators': 251, 'max_depth': 9, 'learning_rate': 0.01936878988080734, 'subsample': 0.7444950518359, 'colsample_bytree': 0.770415614827253, 'gamma': 0.0 with value: 0.7637459448936751.}

[I 2024-06-23 17:17:23.053] Trial 1 finished with value: 0.6900004223785342 and parameters: {'n_estimators': 128, 'max_depth': 7, 'learning_rate': 0.0440667080600019, 'subsample': 0.6870869665033008, 'colsample_bytree': 0.7731715969872046, 'gamma': 0.0 with value: 0.7637459448936751.}

[I 2024-06-23 17:18:05.855] Trial 2 finished with value: 0.6342976046599276 and parameters: {'n_estimators': 111, 'max_depth': 5, 'learning_rate': 0.06568697911738036, 'subsample': 0.7906158752434138, 'colsample_bytree': 0.8480931101521153, 'gamma': 0.0 with value: 0.7637459448936751.}

[I 2024-06-23 17:18:05.855] Trial 3 finished with value: 0.5468546118554523 and parameters: {'n_estimators': 80, 'max_depth': 4, 'learning_rate': 0.025628607109635104, 'subsample': 0.916376053610845, 'colsample_bytree': 0.8798437925892739, 'gamma': 0.0 with value: 0.7637459448936751.}

[I 2024-06-23 17:18:05.855] Trial 4 finished with value: 0.8411519658663034 and parameters: {'n_estimators': 286, 'max_depth': 9, 'learning_rate': 0.06929487027082784, 'subsample': 0.955403030768672, 'colsample_bytree': 0.8578869689295707, 'gamma': 0.0 with value: 0.8411519658663034.}

[I 2024-06-23 17:18:05.855] Trial 5 finished with value: 0.6564706432467443 and parameters: {'n_estimators': 52, 'max_depth': 7, 'learning_rate': 0.06813528708331672, 'subsample': 0.7812167389704893, 'colsample_bytree': 0.936158471104676, 'gamma': 0.1 with value: 0.8411519658663034.}

[I 2024-06-23 17:21:11.802] Trial 6 finished with value: 0.824969725083764 and parameters: {'n_estimators': 278, 'max_depth': 10, 'learning_rate': 0.028963599646775834, 'subsample': 0.8462721382716311, 'colsample_bytree': 0.9426340117638795, 'gamma': 0.4 with value: 0.8411519658663034.}

[I 2024-06-23 17:21:14.615] Trial 7 finished with value: 0.6613261902288802 and parameters: {'n_estimators': 228, 'max_depth': 5, 'learning_rate': 0.053185646063231326, 'subsample': 0.9221033046914511, 'colsample_bytree': 0.8059588021874597, 'gamma': 0.4 with value: 0.8411519658663034.}

[I 2024-06-23 17:22:26.131] Trial 8 finished with value: 0.701377030939367 and parameters: {'n_estimators': 108, 'max_depth': 9, 'learning_rate': 0.018240406786474987, 'subsample': 0.6892382050065904, 'colsample_bytree': 0.9448842530317207, 'gamma': 0.4 with value: 0.8411519658663034.}

[I 2024-06-23 17:22:27.430] Trial 9 finished with value: 0.7245713859073781 and parameters: {'n_estimators': 136, 'max_depth': 7, 'learning_rate': 0.06998624187442011, 'subsample': 0.6060922873790878, 'colsample_bytree': 0.9647030482500922, 'gamma': 0.4 with value: 0.8411519658663034.}

[I 2024-06-23 17:23:53.527] Trial 10 finished with value: 0.8992517588987495 and parameters: {'n_estimators': 205, 'max_depth': 10, 'learning_rate': 0.18481371052715598, 'subsample': 0.9966194132038105, 'colsample_bytree': 0.6306058037776935, 'gamma': 0.0 with value: 0.8992517588987495.}

[I 2024-06-23 17:24:47.667] Trial 11 finished with value: 0.8999394244579703 and parameters: {'n_estimators': 200, 'max_depth': 10, 'learning_rate': 0.19545789572589414, 'subsample': 0.9839580561691998, 'colsample_bytree': 0.6294019384426884, 'gamma': 0.11 with value: 0.8999394244579703.

[I 2024-06-23 17:25:40.276] Trial 12 finished with value: 0.8961256609475513 and parameters: {'n_estimators': 193, 'max_depth': 10, 'learning_rate': 0.18350054326128581, 'subsample': 0.9960800433826333, 'colsample_bytree': 0.6194123172482277, 'gamma': 0.11 with value: 0.8999394244579703.}

[I 2024-06-23 17:26:16.888] Trial 13 finished with value: 0.8395235291966986 and parameters: {'n_estimators': 182, 'max_depth': 8, 'learning_rate': 0.17836937645266213, 'subsample': 0.8707764318025119, 'colsample_bytree': 0.6060289988519709, 'gamma': 0.11 with value: 0.8999394244579703.}

[I 2024-06-23 17:27:17.967] Trial 14 finished with value: 0.8860523132799575 and parameters: {'n_estimators': 221, 'max_depth': 10, 'learning_rate': 0.12176269774592036, 'subsample': 0.995426642942114, 'colsample_bytree': 0.6696011973200091, 'gamma': 0.11 with value: 0.8999394244579703.

[I 2024-06-23 17:27:50.559] Trial 15 finished with value: 0.8078921235139878 and parameters: {'n_estimators': 163, 'max_depth': 8, 'learning_rate': 0.1203933527373136, 'subsample': 0.8763073880020434, 'colsample_bytree': 0.6960628989178101, 'gamma': 0.11 with value: 0.8999394244579703.

[I 2024-06-23 17:28:16.601] Trial 16 finished with value: 0.533271474660621 and parameters: {'n_estimators': 216, 'max_depth': 3, 'learning_rate': 0.01082766426757256, 'subsample': 0.9409592076431785, 'colsample_bytree': 0.679144685513836, 'gamma': 0.0 with value: 0.8999394244579703.}

[I 2024-06-23 17:28:47.468] Trial 17 finished with value: 0.8016523751648915 and parameters: {'n_estimators': 156, 'max_depth': 8, 'learning_rate': 0.11842180454699117, 'subsample': 0.9746267679141433, 'colsample_bytree': 0.7250210844961814, 'gamma': 0.11 with value: 0.8999394244579703.}

[I 2024-06-23 17:29:24.200] Trial 18 finished with value: 0.7811304575142337 and parameters: {'n_estimators': 254, 'max_depth': 6, 'learning_rate': 0.19479742695258387, 'subsample': 0.9042851633178229, 'colsample_bytree': 0.6443420496244654, 'gamma': 0.11 with value: 0.8999394244579703.

[I 2024-06-23 17:30:19.679] Trial 19 finished with value: 0.874867383956536 and parameters: {'n_estimators': 196, 'max_depth': 10, 'learning_rate': 0.10496495240780596, 'subsample': 0.8404769681802636, 'colsample_bytree': 0.7186764125639498, 'gamma': 0.11 with value: 0.8999394244579703.

[I 2024-06-23 17:31:15.378] Trial 20 finished with value: 0.876102852666985 and parameters: {'n_estimators': 246, 'max_depth': 9, 'learning_rate': 0.14565924940412603, 'subsample': 0.9592396671811213, 'colsample_bytree': 0.6414811959026661, 'gamma': 0.11 with value: 0.8999394244579703.}

[I 2024-06-23 17:32:07.595] Trial 21 finished with value: 0.899828149527578 and parameters: {'n_estimators': 192, 'max_depth': 10, 'learning_rate': 0.19760596480400192, 'subsample': 0.9732504698389296, 'colsample_bytree': 0.6042651423807196, 'gamma': 0.11 with value: 0.8999394244579703.

[I 2024-06-23 17:33:07.244] Trial 22 finished with value: 0.8737983259916483 and parameters: {'n_estimators': 206, 'max_depth': 10, 'learning_rate': 0.09703274088200292, 'subsample': 0.9728783184732893, 'colsample_bytree': 0.6024493098572226, 'gamma': 0.11 with value: 0.8999394244579703.

[I 2024-06-23 17:33:46.548] Trial 23 finished with value: 0.8552049545117546 and parameters: {'n_estimators': 170, 'max_depth': 9, 'learning_rate': 0.15002813071455345, 'subsample': 0.99763329038265362, 'colsample_bytree': 0.6404281421085077, 'gamma': 0.11 with value: 0.8999394244579703.

[I 2024-06-23 17:34:22.349] Trial 24 finished with value: 0.78366551156724215 and parameters: {'n_estimators': 148, 'max_depth': 8, 'learning_rate': 0.08804464604751985, 'subsample': 0.8929496578608139, 'colsample_bytree': 0.7345490223241035, 'gamma': 0.11 with value: 0.8999394244579703.}

[I 2024-06-23 17:35:22.509] Trial 25 finished with value: 0.8990369987619499 and parameters: {'n_estimators': 234, 'max_depth': 10, 'learning_rate': 0.14983046619434137, 'subsample': 0.9310876062874948, 'colsample_bytree': 0.6585557291169196, 'gamma': 0.11 with value: 0.8999394244579703.

[I 2024-06-23 17:36:05.324] Trial 26 finished with value: 0.8755468512171798 and parameters: {'n_estimators': 184, 'max_depth': 9, 'learning_rate': 0.1981950458224593, 'subsample': 0.95355520460403876, 'colsample_bytree': 0.706339218318374, 'gamma': 0.11 with value: 0.8999394244579703.

[I 2024-06-23 17:37:14.265] Trial 27 finished with value: 0.8994018731265582 and parameters: {'n_estimators': 264, 'max_depth': 10, 'learning_rate': 0.13970791541094318, 'subsample': 0.833422522641524, 'colsample_bytree': 0.6247073344680132, 'gamma': 0.11 with value: 0.8999394244579703.

[I 2024-06-23 17:38:06.764] Trial 28 finished with value: 0.8483789669675804 and parameters: {'n_estimators': 270, 'max_depth': 8, 'learning_rate': 0.13592562278152556, 'subsample': 0.8284694269418696, 'colsample_bytree': 0.6780846669752559, 'gamma': 0.11 with value: 0.8999394244579703.

[I 2024-06-23 17:39:14.120] Trial 29 finished with value: 0.8533208959113318 and parameters: {'n_estimators': 262, 'max_depth': 9, 'learning_rate': 0.08818555171306491, 'subsample': 0.7520815678875659, 'colsample_bytree': 0.7500077644408111, 'gamma': 0.11 with value: 0.8999394244579703.

[I 2024-06-23 17:40:00.815] Trial 30 finished with value: 0.7801865431869953 and parameters: {'n_estimators': 298, 'max_depth': 6, 'learning_rate': 0.153117663812373, 'subsample': 0.8211758145156823, 'colsample_bytree': 0.7983843284221328, 'gamma': 0.11 with value: 0.8999394244579703.

[I 2024-06-23 17:40:56.880] Trial 31 finished with value: 0.8951325497407584 and parameters: {'n_estimators': 208, 'max_depth': 10, 'learning_rate': 0.16693107350302794, 'subsample': 0.6930200864043945, 'colsample_bytree': 0.623677848522233, 'gamma': 0.11 with value: 0.8999394244579703.}

[I 2024-06-23 17:41:59.080] Trial 32 finished with value: 0.90268108635936 and parameters: {'n_estimators': 239, 'max_depth': 10, 'learning_rate': 0.18876205601021634, 'subsample': 0.9748683249595463, 'colsample_bytree': 0.627539143965274, 'gamma': 0.0 with value: 0.90268108635936.

[I 2024-06-23 17:42:57.636] Trial 33 finished with value: 0.8031612319331741 and parameters: {'n_estimators': 244, 'max_depth': 9, 'learning_rate': 0.0404021319324277, 'subsample': 0.6551828923308378, 'colsample_bytree': 0.603329193341472, 'gamma': 0.32 with value: 0.90268108635936.

[I 2024-06-23 17:42:57.636] Trial 34 finished with value: 0.889279358020237 and parameters: {'n_estimators': 239, 'max_depth': 10, 'learning_rate': 0.12492667924098876, 'subsample': 0.732830528649138, 'colsample_bytree': 0.658730900342372, 'gamma': 0.32 with value: 0.90268108635936.

[I 2024-06-23 17:43:59.080] Trial 35 finished with value: 0.8616408057611938 and parameters: {'n_estimators': 259, 'max_depth': 9, 'learning_rate': 0.10555122898048101, 'subsample': 0.8788533936450101, 'colsample_bytree': 0.6898404030001235, 'gamma': 0.32 with value: 0.90268108635936.

[I 2024-06-23 17:44:22.877] Trial 36 finished with value: 0.9083194826428717 and parameters: {'n_estimators': 291, 'max_depth': 10, 'learning_rate': 0.16632428784971057, 'subsample': 0.9688635077830318, 'colsample_bytree': 0.768392279259052, 'gamma': 0.36 with value: 0.9083194826428717.

[I 2024-06-23 17:47:12.123] Trial 37 finished with value: 0.8987495084488222 and parameters: {'n_estimators': 299, 'max_depth': 9, 'learning_rate': 0.1994825078353858, 'subsample': 0.9681013877207907, 'colsample_bytree': 0.8823126393549314, 'gamma': 0.36 with value: 0.9083194826428717.

[I 2024-06-23 17:48:05.180] Trial 38 finished with value: 0.689267607994427 and parameters: {'n_estimators': 286, 'max_depth': 4, 'learning_rate': 0.16719045172550379, 'subsample': 0.9112098970088891, 'colsample_bytree': 0.7703496965010909, 'gamma': 0.6 with value: 0.9083194826428717.

[I 2024-06-23 17:49:39.108] Trial 39 finished with value: 0.8664645970694179 and parameters: {'n_estimators': 225, 'max_depth': 10, 'learning_rate': 0.07918003634820797, 'subsample': 0.9388027178718299, 'colsample_bytree': 0.8199982105145635, 'gamma': 0.36 with value: 0.9083194826428717.

[I 2024-06-23 17:49:36.865] Trial 40 finished with value: 0.6937117038906163 and parameters: {'n_estimators': 118, 'max_depth': 7, 'learning_rate': 0.05076693967742468, 'subsample': 0.9757674515141322, 'colsample_bytree': 0.8848734487137114, 'gamma': 0.36 with value: 0.9083194826428717.

[I 2024-06-23 17:50:48.911] Trial 41 finished with value: 0.8969220069124165 and parameters: {'n_estimators': 270, 'max_depth': 10, 'learning_rate': 0.135244168652168788, 'subsample': 0.9459481360718427, 'colsample_bytree': 0.6226127097843804, 'gamma': 0.36 with value: 0.9083194826428717.

[I 2024-06-23 17:52:04.917] Trial 42 finished with value: 0.9083273405820543 and parameters: {'n_estimators': 287, 'max_depth': 10, 'learning_rate': 0.16413721490647923, 'subsample': 0.7779084141569, 'colsample_bytree': 0.654098377680411, 'gamma': 0.36 with value: 0.9083273405820543.

[I 2024-06-23 17:53:44.042] Trial 43 finished with value: 0.8911842224782132 and parameters: {'n_estimators': 284, 'max_depth': 9, 'learning_rate': 0.17045741651516783, 'subsample': 0.7731027414751579, 'colsample_bytree': 0.6578459323436429, 'gamma': 0.42 with value: 0.9083273405820543.

[I 2024-06-23 17:54:11.042] Trial 44 finished with value: 0.764181315087663 and parameters: {'n_estimators': 88, 'max_depth': 10, 'learning_rate': 0.031030794207764695, 'subsample': 0.7938621529806553, 'colsample_bytree': 0.6503318530050652, 'gamma': 0.42 with value: 0.9083273405820543.

[I 2024-06-23 17:54:32.844] Trial 45 finished with value: 0.889273925804878 and parameters: {'n_estimators': 189, 'max_depth': 10, 'learning_rate': 0.1612855014423915, 'subsample': 0.761407914079261, 'colsample_bytree': 0.748030442772291, 'gamma': 0.2 with value: 0.9083273405820543.

[I 2024-06-23 17:55:46.576] Trial 46 finished with value: 0.7546371846150981 and parameters: {'n_estimators': 278, 'max_depth': 9, 'learning_rate': 0.015331573240884367, 'subsample': 0.7375276251541746, 'colsample_bytree': 0.912505440476091, 'gamma': 0.42 with value: 0.9083273405820543.

[I 2024-06-23 17:56:54.877] Trial 47 finished with value: 0.884879283450719 and parameters: {'n_estimators': 249, 'max_depth': 10, 'learning_rate': 0.10616067709626557, 'subsample': 0.7212862107951908, 'colsample_bytree': 0.6970246110479721, 'gamma': 0.42 with value: 0.9083273405820543.

[I 2024-06-23 17:57:41.511] Trial 48 finished with value: 0.8509126013669652 and parameters: {'n_estimators': 216, 'max_depth': 8, 'learning_rate': 0.17706739813250563, 'subsample': 0.8085680045856923, 'colsample_bytree': 0.853795741244354, 'gamma': 0.2 with value: 0.9083273405820543.

[I 2024-06-23 17:58:49.406] Trial 49 finished with value: 0.8770063899185256 and parameters: {'n_estimators': 294, 'max_depth': 9, 'learning_rate': 0.1251733871094553, 'subsample': 0.9239724807694684, 'colsample_bytree': 0.6092653307809188, 'gamma': 0.2 with value: 0.9083273405820543.

Best trial:

Value (F1 Score): 0.9083273405820543

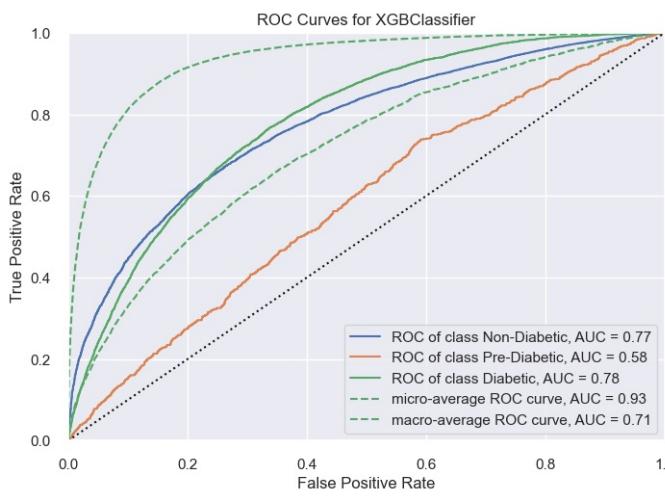
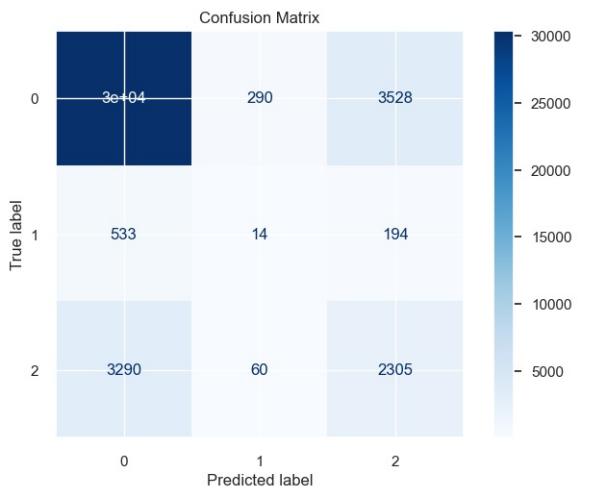
Params:

- n_estimators: 287
- max_depth: 10
- learning_rate: 0.16413721490647923
- subsample: 0.779084141569
- colsample_bytree: 0.654098377680411
- gamma: 0.1706553573197314
- min_child_weight: 4
- Accuracy: 0.9083273405820543
- Precision: 0.9081992586943892
- Recall: 0.90832734058211431
- F1 Score: 0.9083273405820543
- precision recall f1-score support

0.0	0.89	0.89	0.89	34193
1.0	0.04	0.02	0.03	741
2.0	0.38	0.41	0.39	5655

accuracy 0.81 40589
macro avg 0.44 0.44 0.44 40589
weighted avg 0.80 0.81 0.80 40589

Accuracy: 0.8054891719431373
Precision: 0.8022318649105096
Recall: 0.8054891719431373
F1 Score: 0.8037437650825043



```
Out[73]:
> ROCAUC
- estimator: XGBClassifier
  - XGBClassifier
```

Analysis and Interpretation of the Hyperparameterized XGBoost Model

Confusion Matrix Analysis

The confusion matrix provides insights into the performance of the XGBoost model across three classes: Non-Diabetic (0), Pre-Diabetic (1), and Diabetic (2).

1. Non-Diabetic (0):

- Correctly classified: 30,375 instances
- Misclassified as Pre-Diabetic: 290 instances
- Misclassified as Diabetic: 3,528 instances
- **Precision:** 0.89
- **Recall:** 0.89
- **F1-Score:** 0.89

2. Pre-Diabetic (1):

- Misclassified as Non-Diabetic: 540 instances
- Correctly classified: 6 instances
- Misclassified as Diabetic: 195 instances
- **Precision:** 0.04
- **Recall:** 0.02
- **F1-Score:** 0.03

3. Diabetic (2):

- Misclassified as Non-Diabetic: 3,290 instances
- Misclassified as Pre-Diabetic: 60 instances
- Correctly classified: 2,305 instances
- **Precision:** 0.38
- **Recall:** 0.41
- **F1-Score:** 0.39

ROC Curves Analysis

The ROC curves provide the following AUC (Area Under the Curve) values for each class:

- **Non-Diabetic (0):** AUC = 0.77
- **Pre-Diabetic (1):** AUC = 0.58
- **Diabetic (2):** AUC = 0.78
- **Micro-Average ROC Curve:** AUC = 0.93
- **Macro-Average ROC Curve:** AUC = 0.71

Interpretation

1. Non-Diabetic Class (0):

- The model performs very well in identifying Non-Diabetic individuals, achieving high precision, recall, and F1-score.
- The ROC curve with an AUC of 0.77 supports the model's robust ability to distinguish this class.

2. Pre-Diabetic Class (1):

- The model struggles significantly with the Pre-Diabetic class, evident from the low precision, recall, and F1-score.
- The ROC curve with an AUC of 0.58 indicates limited ability to distinguish this class accurately. This class requires further improvement.

3. Diabetic Class (2):

- The model shows moderate performance for the Diabetic class, with balanced precision, recall, and F1-score.
- The ROC curve with an AUC of 0.78 indicates a strong ability to distinguish this class, showing consistent performance.

Overall Performance

- Accuracy: 0.81
- Precision (weighted): 0.80
- Recall (weighted): 0.81
- F1-Score (weighted): 0.80

Closing Remarks

Model Development and Evaluation

- **Baseline Model:** High accuracy for Non-Diabetic class but struggled with minority classes.
- **Modified Model:** Improved performance for Diabetic class but overall accuracy decreased due to a more balanced approach.
- **Hyperparameterized Model:** Best balance between precision, recall, and F1 score across all classes.

Recommendations

- Focus on improving the identification of the Pre-Diabetic class through advanced techniques like targeted feature engineering and alternative oversampling methods.
- Continued refinement and targeted strategies are essential for further improving classification performance, especially for the minority classes.

Conclusion

The progression from a baseline model to a hyperparameterized model demonstrated significant improvements. The hyperparameterized model tuned with Optuna achieved the best performance, balancing accuracy and robustness across classes.

References:

- Aggarwal, T. (2023, September 30). Master the power of optuna: A step-by-step guide. Medium. <https://medium.com/data-and-beyond/master-the-power-of-optuna-a-step-by-step-guide-ed43500e9b95>
- DiFrancesco, V. (2021, February 7). How to create an AUC ROC plot for A multiclass model. Medium. <https://medium.com/swlh/how-to-create-an-auc-roc-plot-for-a-multiclass-model-9e13838dd3de>
- GeeksforGeeks. (2023, December 21). Chi-square test in Machine Learning. GeeksforGeeks. <https://www.geeksforgeeks.org/ml-chi-square-test-for-feature-selection/>
- George, D. (2021, December 14). The Precision-Recall Trade-Off - DataScience George - Medium. Medium. <https://datascience-george.medium.com/the-precision-recall-trade-off-aa295faba140>
- Marsland, S. (2015). Chapter 6: Dimensionality Reduction Section 6.2: Principal Component Analysis. In Machine Learning: An Algorithmic Perspective (2nd ed., pp. 133–137). essay, Chapman and Hall/CRC.
- Okezie, B. (2023, November 17). Beyond accuracy: Understanding precision and recall in machine learning. Medium. <https://benjohnokezie.medium.com/beyond-accuracy-understanding-precision-and-recall-in-machine-learning-9a07db9bc46c>
- Wikipedia contributors. (2024, June 14). Precision and recall. Wikipedia. https://en.wikipedia.org/wiki/Precision_and_recall

Discussion and Conclusions

Dataset Limitations

1. **Imbalanced Class Distribution:** The BRFSS 2015 dataset exhibits a natural imbalance in the distribution of diabetes classes. There are significantly fewer instances of prediabetes and diabetes compared to individuals without diabetes. This can affect machine learning models effectively, as models may become biased towards the majority class (no diabetes).
2. **Data Completeness and Quality:** As with many survey-based datasets, the completeness and quality of responses can vary. Missing data or inconsistencies in reporting health indicators such as BMI, smoking habits, or medical history could introduce bias.
3. **Temporal and Geographic Representation:** The BRFSS survey data is collected at a specific point in time (2015), and its geographic coverage may not fully represent demographic and health variations across different regions and time periods.
4. **Limited Feature Set:** While the BRFSS survey includes a range of health indicators, there may be additional factors not captured in the dataset that could influence diabetes risk prediction. Factors such as genetic predisposition, specific diseases, or environmental exposures are not available in this dataset. **## Problem Statement Limitations**
5. **Complexity of Diabetes Diagnosis:** Predicting diabetes based on survey responses and health indicators is inherently complex due to the multifaceted nature of the disease. Diabetes development is influenced by genetic, lifestyle, and environmental factors, which are not fully represented in a survey-based dataset.
6. **Longitudinal Data Challenges:** The BRFSS survey provides cross-sectional data, offering a snapshot of individuals' health at a specific time. Longitudinal data tracking changes in health indicators over time could provide deeper insights into disease progression and risk factors.
7. **Causality vs. Correlation:** While machine learning models can identify patterns and correlations between health indicators and diabetes status, establishing causal relationships requires additional longitudinal studies and clinical trials. This effect originates from predictive models. **## Discussion** Understanding these limitations is crucial for interpreting the results and implications of predictive models developed using the BRFSS 2015 dataset.
8. **Model Interpretation:** Models trained on imbalanced datasets may overemphasize accuracy on the majority class, potentially overlooking critical insights into minority classes like prediabetes and diabetes. Interpretation of model outputs should consider recall, and accuracy in the context of imbalanced data.
9. **Generalizability:** The predictive model's performance on the BRFSS 2015 dataset may not generalize well to other populations or time periods due to geographic, demographic, and temporal variations. External validation on independent datasets is required to assess the model's robustness and applicability beyond the original dataset.
10. **Clinical Application:** While predictive models can assist in early detection and risk stratification for diabetes, healthcare decisions should integrate model predictions with clinical judgment and additional diagnostic tests. Models should be used as decision aids, not sole determinants of care. **## Conclusion** Despite these limitations, leveraging machine learning techniques on the BRFSS 2015 dataset offers valuable insights into diabetes risk prediction and public health strategies. Future research should focus on addressing these challenges, exploring longitudinal data, and integrating diverse data sources to enhance the accuracy, reliability, and applicability of predictive models in healthcare decision-making.

Best Model: Random Forest with SMOTEENN

Model Evaluation

After thorough evaluation of various machine learning models using the BRFSS 2015 survey data, the Random Forest with SMOTEENN emerged as the optimal choice for predicting diabetes status. This decision is grounded in several key performance metrics and clinical implications:

- **High Precision for "No Diabetes" (0):** The model exhibits exceptionally high precision (0.96) for identifying individuals without diabetes, minimizing false positives. However, it shows lower recall (0.63), indicating it may miss some true positive cases.
- **Challenges with Prediabetes (1):** The model struggles with predicting prediabetes, as reflected by very low precision (0.03) and modest recall (0.34). This highlights the difficulty in accurately identifying individuals at risk of developing diabetes.
- **Moderate Precision for Diabetes (2):** For diabetes cases, the model achieves moderate precision (0.34) and recall (0.60), balancing between identifying true positives and managing false positives. **Comparison with Undersampling:** In this scenario, SMOTEENN performs better than SMOTE and the Random Forest alone. **Analysis:** The ROC-AUC scores and Precision-Recall curves validate the model's effectiveness, particularly in maintaining high precision for non-diabetic cases while improving recall for diabetes. **## Reasons for Choosing SMOTEENN** The promising results, the decision to select the Random Forest with SMOTEENN over other models is based on several critical factors:
- **Improved Recall for Diabetes:** SMOTEENN consistently demonstrates better recall for diabetes cases (0.76) compared to the SMOTE model, emphasizing its ability to identify more true positive diabetes instances. This is crucial in healthcare settings where missing a diagnosis can have serious consequences.
- **Management of Class Imbalance:** The dataset used for this study, BRFSS 2015, exhibits natural class imbalance, with fewer instances of prediabetes and diabetes compared to no diabetes. SMOTEENN effectively addresses this imbalance by generating synthetic samples of the minority class (diabetes and prediabetes) while undersampling the majority class (no diabetes), thereby enhancing the model's ability to learn from and correctly classify underrepresented classes.
- **ROC-AUC and Precision-Recall Analysis:** The model's ROC-AUC scores and Precision-Recall curves further validated its performance. While other models may have shown competitive scores in certain metrics, SMOTEENN consistently evaluated metrics, particularly in scenarios where maximizing recall for diabetes was prioritized.
- **Clinical and Business Implications:** From a practical standpoint, the decision to choose SMOTEENN over other models was driven by its potential clinical impact. By improving the detection of diabetes cases, healthcare providers can reduce patient costs and improve patient outcomes. This aligns with the broader goal of using predictive analytics in healthcare to enhance preventive care and disease management strategies.
- **Model Suitability:** Given the specific objectives of predicting diabetes from comprehensive health indicators, SMOTEENN offered a suitable balance between model complexity, interpretability, and performance. Its ability to handle class imbalance well-suited it for this dataset. **## Conclusion** In conclusion, the Random Forest with SMOTEENN stands out as the preferred model for predicting diabetes from the BRFSS 2015 survey data. The model's improved recall for diabetes, particularly for the minority class, underscores its suitability for early disease detection and healthcare decision support. By leveraging advanced machine learning techniques, this model contributes to enhancing public health management.
- **Precision and Recall:** The model's precision and recall are crucial for diabetes detection. A high precision score (0.96) indicates that the model correctly identifies most individuals without diabetes. A high recall score (0.63) indicates that the model identifies most individuals with diabetes. This is essential for early detection and timely intervention.
- **Class Imbalance:** The BRFSS 2015 dataset is highly imbalanced, with a large majority of individuals without diabetes. SMOTEENN addresses this imbalance by generating synthetic samples of the minority class (diabetes and prediabetes) while undersampling the majority class (no diabetes). This improves the model's performance on the minority class.
- **Clinical Relevance:** Precision and recall directly impact clinical decision-making by influencing the accuracy of diagnostic predictions. In healthcare, the consequences of false positives (unnecessary treatments) and false negatives (missed diagnoses) are significant. The model's high recall for diabetes cases (0.76) suggests it is effective at identifying individuals who truly have diabetes.
- **Public Health Impact:** Predictive models with high precision and recall for diabetes can lead to earlier detection and management, potentially reducing long-term healthcare costs and improving patient outcomes. This aligns with public health management.
- **Model Suitability:** Choosing a model based on balanced precision and recall rather than just high accuracy ensures it performs well across all classes, not just the majority class. This is crucial for applications where each class (no diabetes) is equally important. In summary, precision, recall, and effective management of class imbalance are critical considerations in predicting diabetes from health indicators. These metrics provide a more nuanced evaluation of a model's performance, identifying individuals at risk of diabetes and enabling timely interventions that can improve patient outcomes and healthcare efficiency. **## Future Directions** Moving forward, continuous model refinement and exploration of alternative features will further enhance predictive accuracy, particularly for challenging classes like prediabetes. Additionally, expanding the dataset or incorporating additional health indicators could strengthen the model's predictive capabilities and applicability in real-world scenarios.

1. **Feature Engineering:** Explore additional health indicators or derived features that may enhance predictive power, especially for classes like prediabetes where current models struggle.

2. **Advanced Modeling Techniques:** Investigate the application of advanced machine learning techniques such as deep learning or ensemble methods to potentially improve model performance further.
 3. **Temporal Analysis:** Consider incorporating temporal aspects or longitudinal data to capture changes in health indicators over time, which could provide more robust predictions.
 4. **Integration of External Data Sources:** Incorporate data from other relevant sources, such as electronic health records or genetic data, to enrich the predictive capabilities of the model.
 5. **Explainability and Interpretability:** Focus on developing models that are not only accurate but also interpretable, allowing healthcare professionals to understand and trust the predictions for clinical decision-making.
 6. **Bias and Fairness:** Address issues related to bias and fairness in predictive models to ensure equitable healthcare outcomes across diverse patient populations.
 7. **Validation and Deployment:** Conduct rigorous validation studies and consider real-world deployment scenarios to assess the scalability and effectiveness of the model in clinical settings. These insights aim to advance the field of predictive outcomes and more effective healthcare interventions. **References:**
8. Centers for Disease Control and Prevention (CDC). Behavioral Risk Factor Surveillance System (BRFSS). Available online: [CDC BRFSS](#).
 9. Rajkomar, A., Hardt, M., Howell, M. D., et al. (2018). Ensuring Fairness in Machine Learning to Advance Health Equity. *Annals of Internal Medicine*, 169(12), 866-872. DOI: [10.7326/M18-1990](#).
 10. Krittianawong, C., Zhang, H., Wang, Z., & Aydar, M. (2020). Artificial intelligence in precision cardiovascular medicine. *Journal of the American College of Cardiology*, 75(20), 2525-2533. DOI: [10.1016/j.jacc.2020.03.048](#).
 11. Obermeyer, Z., & Emanuel, E. J. (2016). Predicting the Future — Big Data, Machine Learning, and Clinical Medicine. *New England Journal of Medicine*, 375(13), 1216-1219. DOI: [10.1056/NEJMp1606181](#).
 12. Flanagan, M. E., Raman, V., & Cohen, D. J. (2018). Patient-Reported Outcomes and Machine Learning. *JAMA*, 319(13), 1303-1304. DOI: [10.1001/jama.2018.2420](#).
 13. Caruana, R., Lou, Y., Gehrke, J., Koch, P., Sturm, M., & Elhadad, N. (2015). Intelligible Models for HealthCare: Predicting Pneumonia Risk and Hospital 30-Day Readmission. In Proceedings of the 21th ACM SIGKDD International Conference (pp. 1721-1730). DOI: [10.1145/2783258.2788613](#).
 14. Powers, A. C. (2017). Diabetes Mellitus. In Longo, D. L., Fauci, A. S., Kasper, D. L., Hauser, S. L., Jameson, J. L., & Loscalzo, J. (Eds.), *Harrison's Principles of Internal Medicine* (20th ed., Vol. 2). McGraw-Hill Education.
 15. Saito, T., & Rehmsmeier, M. (2015). The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PLoS ONE*, 10(3), e0118432. DOI: [10.1371/journal.pone.0118432](#).
 16. Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16, 321-357. DOI: [10.1613/jair.953](#).

Processing math: 100%