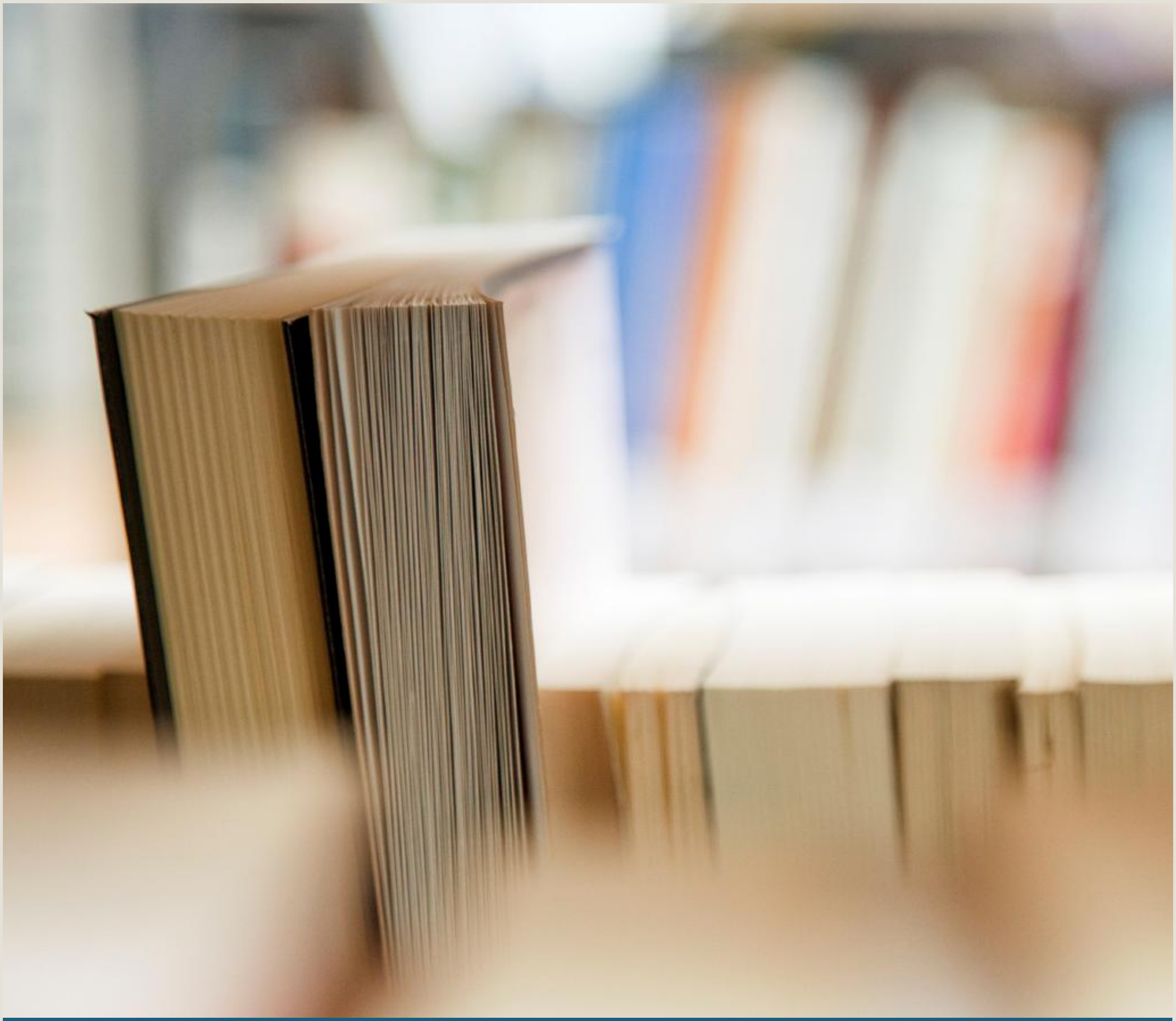


Indicators of Heart Disease

Heart Disease Prediction Using Machine Learning

Nana Firdausi Hassan

Table of Contents

PROJECT TITLE

Indicators of Heart Disease

DATASET

The Indicators of Heart Disease dataset, curated by Kamil Pytlak and publicly available on Kaggle, is derived from the Centers for Disease Control and Prevention's (CDC) Behavioral Risk Factor Surveillance System (BRFSS). The BRFSS is an extensive health-related telephone survey system that collects data annually from adults across all 50 U.S. states regarding their health behaviors, chronic conditions, and preventive health practices.

This specific dataset comprises responses from over 319,000 individuals collected in 2020. While the original BRFSS dataset includes 279 variables, this version has been streamlined to include 18 features that are most relevant for the prediction of heart disease. These variables encompass both clinical and behavioral indicators such as body mass index (BMI), smoking and alcohol consumption, physical and mental health status, and the presence of chronic conditions including diabetes, asthma, and kidney disease. Additionally, the dataset includes demographic variables such as age category, sex, and race.

Due to its breadth and relevance, the dataset serves as a robust foundation for machine learning and statistical modeling efforts aimed at predicting the likelihood of heart disease in individuals. It has been widely used in academic and applied research to develop classification models employing algorithms such as logistic regression, support vector machines, random forests, and neural networks. Several notable projects have utilized the dataset to achieve high-performance metrics, including ROC/AUC scores exceeding 84%.

The dataset is accessible in CSV format on the Kaggle platform, facilitating its use in various data science environments. It provides a valuable resource for researchers, data scientists, and public health analysts interested in predictive modeling and the analysis of key health indicators related to cardiovascular disease.

Here is the link to the data:
Link

STEP1

Objectives

The objective of this project is to analyze the relationship between various health conditions, such as diabetes, stroke, asthma, kidney disease, skin cancer, and alcohol consumption, and the likelihood of heart disease.

Another goal is to preprocess and clean the dataset for machine learning by handling missing data, encoding categorical variables, addressing class imbalance, and standardizing features to ensure the dataset is properly prepared for modeling.

The project aims to build and evaluate multiple machine learning models, including Logistic Regression, Random Forest, Gradient Boosting, XGBoost, and LightGBM, and compare their performance in terms of accuracy, precision, recall, and F1-score.

To improve model performance, the project seeks to address class imbalance by applying resampling techniques such as SMOTEENN (Synthetic Minority Over-sampling Technique + Edited Nearest Neighbors), which balances the dataset, particularly for the minority class (heart disease).

The project also aims to assess the impact of feature selection and dimensionality reduction on model performance by using Principal Component Analysis (PCA) to reduce the number of features while preserving the dataset's variance.

Finally, the objective is to provide insights and recommendations based on the machine learning model results, interpret the outcomes, and identify areas for further improvement in heart disease prediction, along with suggestions for future work.

Problem Statement

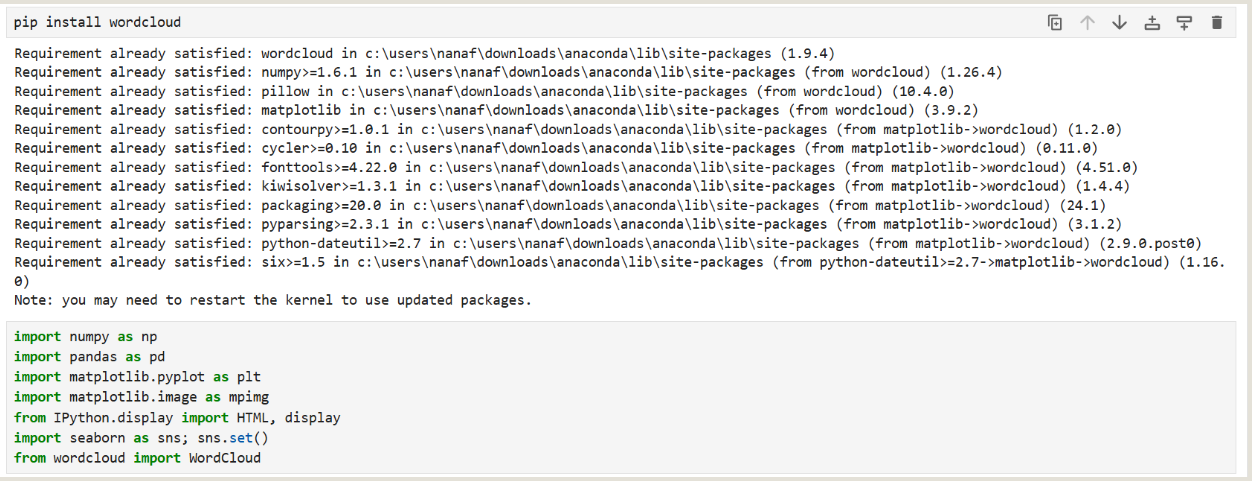
Heart disease remains one of the leading causes of death globally, with lifestyle factors and chronic conditions playing a significant role in its development. Despite the availability of large-scale health data, early identification of individuals at risk for heart disease remains a challenge, particularly in resource-limited settings where access to clinical diagnostics may be restricted. There is a pressing need for accessible, data-driven tools that can support early detection and preventive healthcare interventions by leveraging non-invasive, self-reported health indicators.

Proposed Solution

This project proposes the development of a machine learning-based classification model that can predict the likelihood of heart disease using key personal health indicators from the CDC’s BRFSS dataset. By analyzing variables such as BMI, smoking habits, mental and physical health, and chronic conditions, the model aims to identify individuals at risk with high accuracy. The resulting predictive tool could serve as a low-cost, scalable support system for public health screening, enabling earlier interventions and potentially reducing the burden of heart disease through timely lifestyle and medical guidance.

STEP2

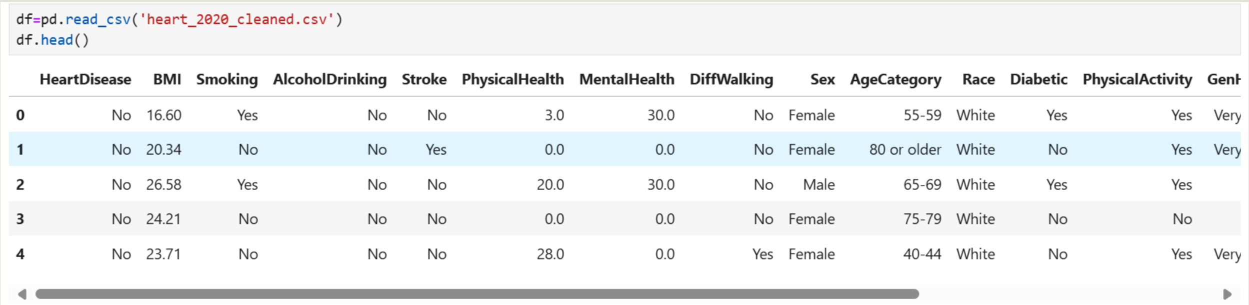
Get the data



```
pip install wordcloud
```

```
Requirement already satisfied: wordcloud in c:\users\nanaf\downloads\anaconda\lib\site-packages (1.9.4)
Requirement already satisfied: numpy>=1.6.1 in c:\users\nanaf\downloads\anaconda\lib\site-packages (from wordcloud) (1.26.4)
Requirement already satisfied: pillow in c:\users\nanaf\downloads\anaconda\lib\site-packages (from wordcloud) (10.4.0)
Requirement already satisfied: matplotlib in c:\users\nanaf\downloads\anaconda\lib\site-packages (from wordcloud) (3.9.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\nanaf\downloads\anaconda\lib\site-packages (from matplotlib->wordcloud) (1.2.0)
Requirement already satisfied: cycler>=0.10 in c:\users\nanaf\downloads\anaconda\lib\site-packages (from matplotlib->wordcloud) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\nanaf\downloads\anaconda\lib\site-packages (from matplotlib->wordcloud) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\nanaf\downloads\anaconda\lib\site-packages (from matplotlib->wordcloud) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\nanaf\downloads\anaconda\lib\site-packages (from matplotlib->wordcloud) (24.1)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\nanaf\downloads\anaconda\lib\site-packages (from matplotlib->wordcloud) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\nanaf\downloads\anaconda\lib\site-packages (from matplotlib->wordcloud) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in c:\users\nanaf\downloads\anaconda\lib\site-packages (from python-dateutil>=2.7->matplotlib->wordcloud) (1.16.
0)
Note: you may need to restart the kernel to use updated packages.
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from IPython.display import HTML, display
import seaborn as sns; sns.set()
from wordcloud import WordCloud
```



```
df=pd.read_csv('heart_2020_cleaned.csv')
df.head()
```

	HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	DiffWalking	Sex	AgeCategory	Race	Diabetic	PhysicalActivity	GenH
0	No	16.60	Yes	No	No	3.0	30.0	No	Female	55-59	White	Yes	Yes	Very
1	No	20.34	No	No	Yes	0.0	0.0	No	Female	80 or older	White	No	Yes	Very
2	No	26.58	Yes	No	No	20.0	30.0	No	Male	65-69	White	Yes	Yes	
3	No	24.21	No	No	No	0.0	0.0	No	Female	75-79	White	No	No	
4	No	23.71	No	No	No	28.0	0.0	Yes	Female	40-44	White	No	Yes	Very

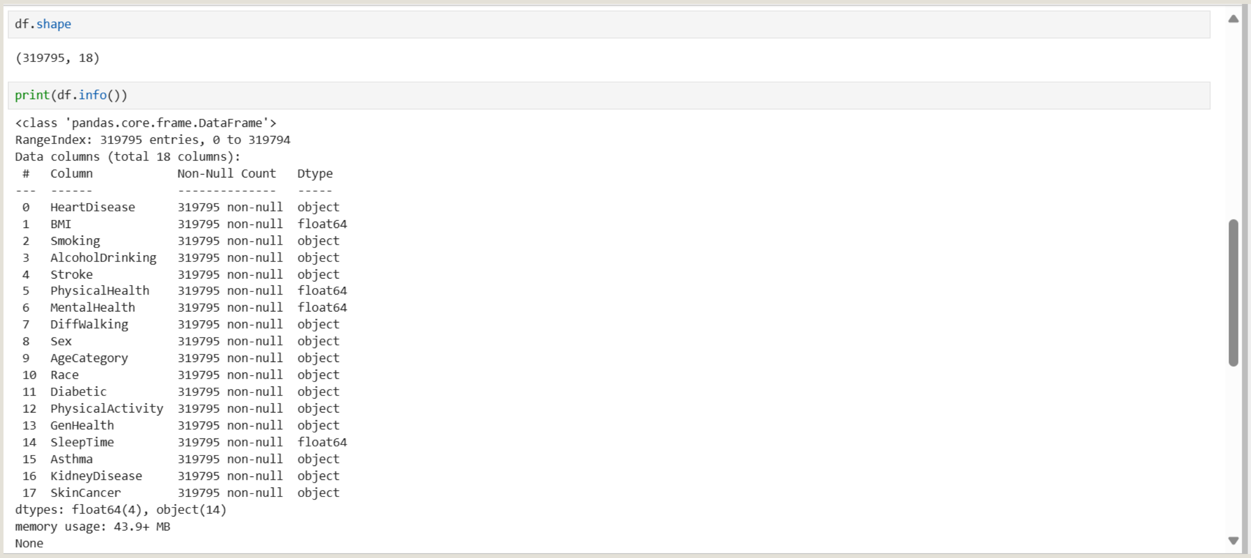
To begin the research, I first imported a set of essential libraries: numpy for numerical operations, pandas for data manipulation, matplotlib and seaborn for data visualization, IPython.display for enhanced output display, and wordcloud for generating word clouds.

After importing the necessary packages, I loaded the dataset heart_2020_cleaned.csv using the read_csv() function from pandas. To confirm that the file was loaded successfully and to get an initial view of the data, I used the .head() function, which displayed the first five rows of the dataset.

STEP 3

Analyze the data to uncover insights

Understanding the dataset

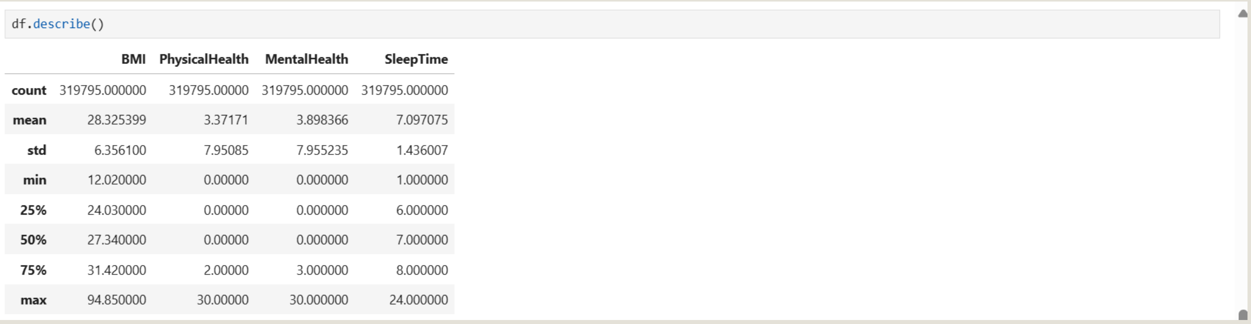


```
df.shape
```

```
(319795, 18)
```

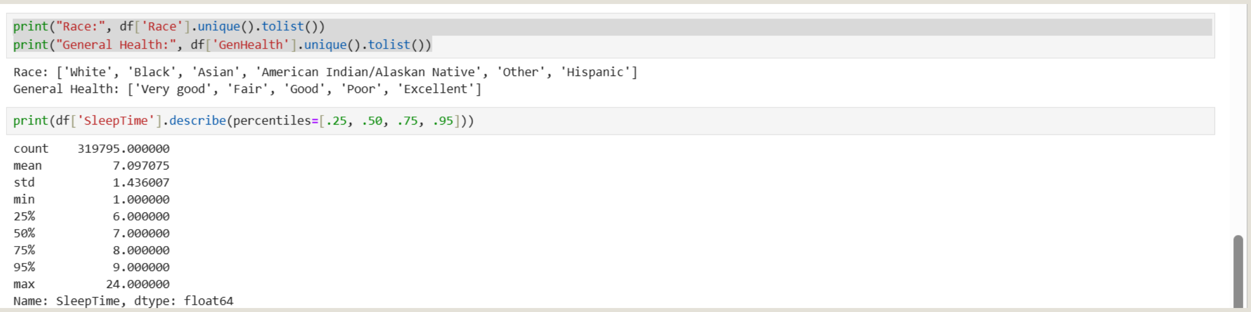
```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 319795 entries, 0 to 319794
Data columns (total 18 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   HeartDisease      319795 non-null  object 
 1   BMI               319795 non-null  float64
 2   Smoking           319795 non-null  object 
 3   AlcoholDrinking   319795 non-null  object 
 4   Stroke            319795 non-null  object 
 5   PhysicalHealth    319795 non-null  float64
 6   MentalHealth      319795 non-null  float64
 7   DiffWalking       319795 non-null  object 
 8   Sex               319795 non-null  object 
 9   AgeCategory       319795 non-null  object 
 10  Race              319795 non-null  object 
 11  Diabetic          319795 non-null  object 
 12  PhysicalActivity  319795 non-null  object 
 13  GenHealth         319795 non-null  object 
 14  SleepTime         319795 non-null  float64
 15  Asthma            319795 non-null  object 
 16  KidneyDisease     319795 non-null  object 
 17  SkinCancer        319795 non-null  object 
dtypes: float64(4), object(14)
memory usage: 43.9+ MB
None
```



```
df.describe()
```

	BMI	PhysicalHealth	MentalHealth	SleepTime
count	319795.000000	319795.00000	319795.000000	319795.000000
mean	28.325399	3.37171	3.898366	7.097075
std	6.356100	7.95085	7.955235	1.436007
min	12.020000	0.00000	0.000000	1.000000
25%	24.030000	0.00000	0.000000	6.000000
50%	27.340000	0.00000	0.000000	7.000000
75%	31.420000	2.00000	3.000000	8.000000
max	94.850000	30.00000	30.000000	24.000000



```
print("Race:", df['Race'].unique().tolist())
print("General Health:", df['GenHealth'].unique().tolist())
```

```
Race: ['White', 'Black', 'Asian', 'American Indian/Alaskan Native', 'Other', 'Hispanic']
General Health: ['Very good', 'Fair', 'Good', 'Poor', 'Excellent']
```

```
print(df['SleepTime'].describe(percentiles=[.25, .50, .75, .95]))
```

```
count    319795.000000
mean          7.097075
std           1.436007
min           1.000000
25%           6.000000
50%           7.000000
75%           8.000000
95%           9.000000
max          24.000000
Name: SleepTime, dtype: float64
```

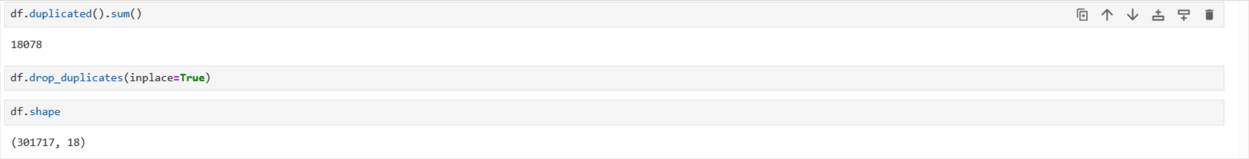
I began by using df.shape to determine that the dataset contains 319,795 rows and 18 columns. To examine the data structure, I used df.info(), which showed that all columns have complete entries with no missing values. The dataset includes 4 numeric columns (BMI, PhysicalHealth, MentalHealth, and SleepTime) and 14 categorical columns.

I then used df.describe() to view summary statistics for the numeric features. This revealed potential outliers, such as BMI values up to 94.85 and SleepTime values as high as 24 hours.

I explored categorical variables using .unique().tolist() for Race and GenHealth, confirming consistent and expected category values. Finally, I used describe(percentiles=[.25, .50, .75, .95]) on SleepTime to assess its distribution, identifying 9 hours as the 95th percentile and flagging the maximum value of 24 as a potential outlier.

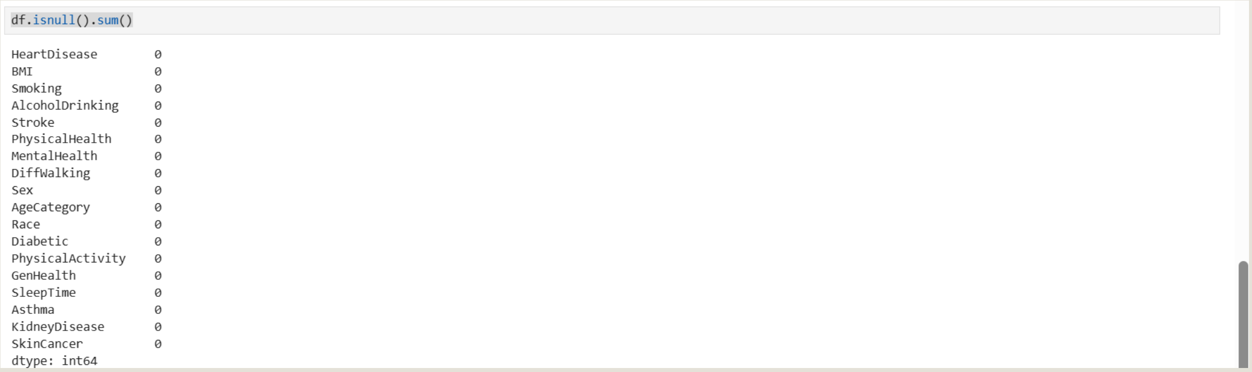
STEP 4

Data Cleaning

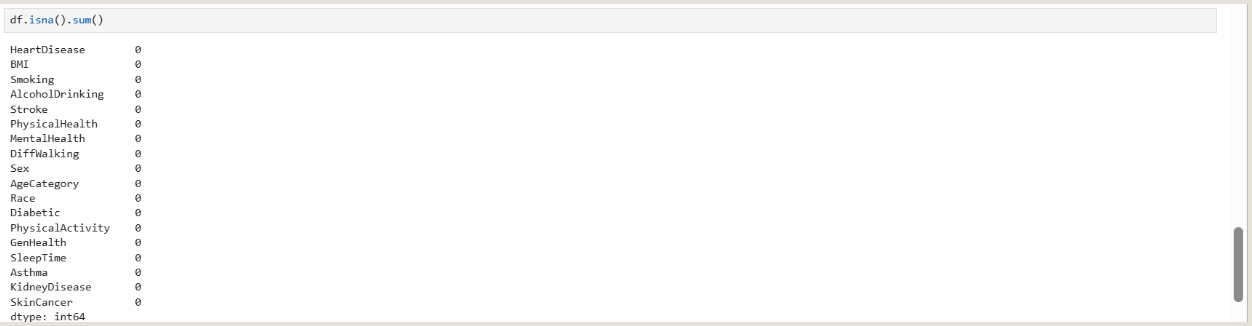


```
df.duplicated().sum()

18078

df.drop_duplicates(inplace=True)

df.shape

(301717, 18)
```

To ensure data quality and eliminate redundancy, I checked for duplicate records in the dataset using the duplicated().sum() function, which revealed 18,078 duplicate entries. These duplicates were removed using the drop_duplicates() method with inplace = True to apply the changes directly to the dataset. After the removal, I confirmed the new dataset shape with df.shape, which showed that 301,717 rows and 18 columns remained. This step helped streamline the dataset for more accurate analysis.

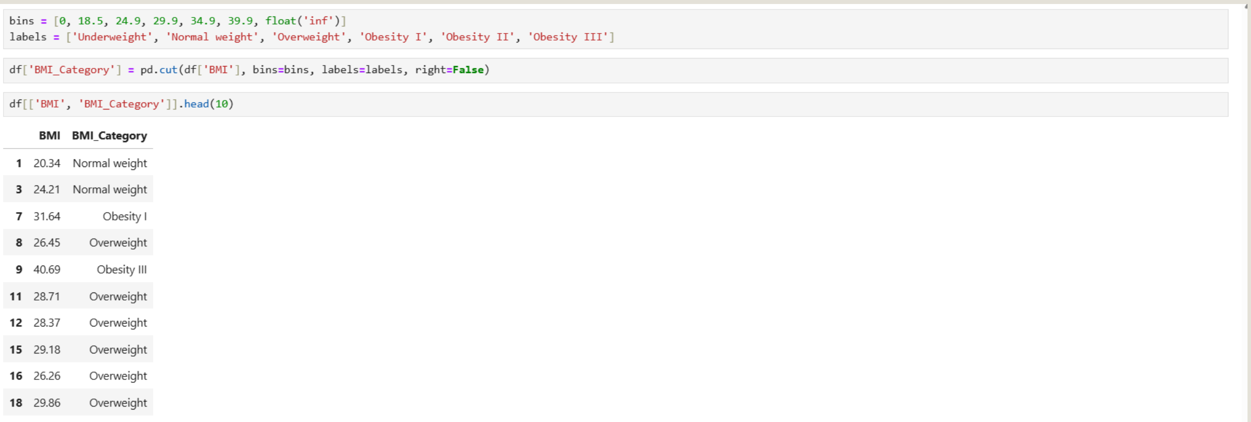


```
df.isnull().sum()

HeartDisease        0
BMI                 0
Smoking             0
AlcoholDrinking     0
Stroke              0
PhysicalHealth      0
MentalHealth        0
DiffWalking         0
Sex                 0
AgeCategory         0
Race                0
Diabetic            0
PhysicalActivity    0
GenHealth           0
SleepTime           0
Asthma              0
KidneyDisease       0
SkinCancer          0
dtype: int64
```



```
df.isna().sum()

HeartDisease        0
BMI                 0
Smoking             0
AlcoholDrinking     0
Stroke              0
PhysicalHealth      0
MentalHealth        0
DiffWalking         0
Sex                 0
AgeCategory         0
Race                0
Diabetic            0
PhysicalActivity    0
GenHealth           0
SleepTime           0
Asthma              0
KidneyDisease       0
SkinCancer          0
dtype: int64
```

To confirm completeness of data, I used both df.isnull().sum(), and df.isna().sum(), which returned zero missing values across all columns.

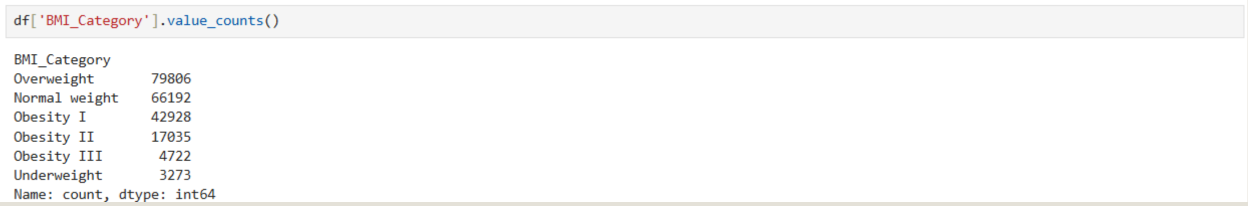


```
bins = [0, 18.5, 24.9, 29.9, 34.9, 39.9, float('inf')]
labels = ['Underweight', 'Normal weight', 'Overweight', 'Obesity I', 'Obesity II', 'Obesity III']
```

```
df['BMI_Category'] = pd.cut(df['BMI'], bins=bins, labels=labels, right=False)
```

```
df[['BMI', 'BMI_Category']].head(10)
```

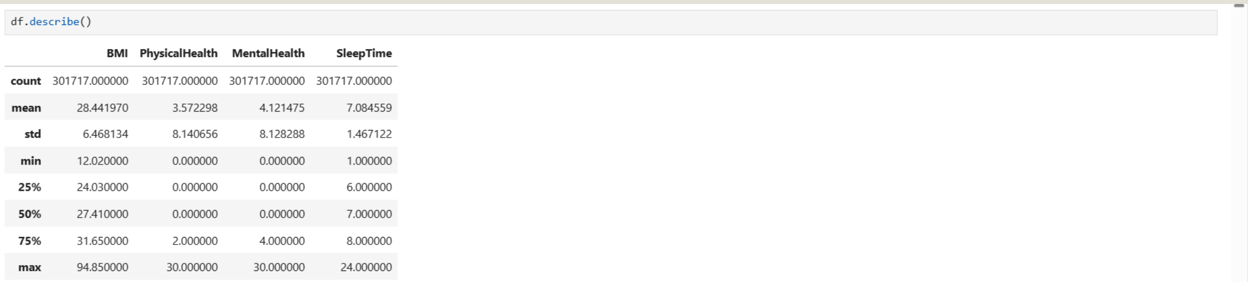
	BMI	BMI_Category
1	20.34	Normal weight
3	24.21	Normal weight
7	31.64	Obesity I
8	26.45	Overweight
9	40.69	Obesity III
11	28.71	Overweight
12	28.37	Overweight
15	29.18	Overweight
16	26.26	Overweight
18	29.86	Overweight



```
df['BMI_Category'].value_counts()
```

```
BMI_Category
Overweight       79806
Normal weight    66192
Obesity I        42928
Obesity II       17035
Obesity III       4722
Underweight       3273
Name: count, dtype: int64
```

I categorized individuals based on their BMI by creating a new column, BMI_Category. Using defined bin ranges and labels, I classified BMI values into six standard categories: Underweight, Normal weight, Overweight, Obesity I, Obesity II, and Obesity III. This was done using pd.cut() with specified bin edges. The first few entries show that the BMI values were correctly assigned to their respective categories, enabling clearer analysis of weight distribution across the dataset.



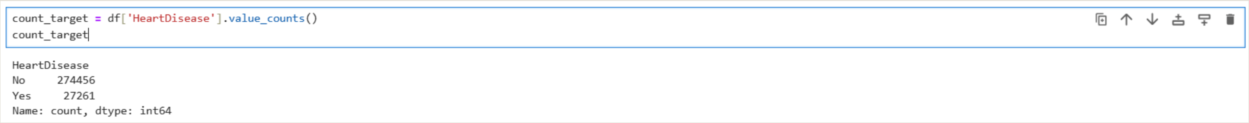
```
df.describe()
```

	BMI	PhysicalHealth	MentalHealth	SleepTime
count	301717.000000	301717.000000	301717.000000	301717.000000
mean	28.441970	3.572298	4.121475	7.084559
std	6.468134	8.140656	8.128288	1.467122
min	12.020000	0.000000	0.000000	1.000000
25%	24.030000	0.000000	0.000000	6.000000
50%	27.410000	0.000000	0.000000	7.000000
75%	31.650000	2.000000	4.000000	8.000000
max	94.850000	30.000000	30.000000	24.000000

After removing 18,078 duplicate rows, the dataset now contains 301,717 unique records. The average BMI is 28.44, with values ranging from 12.02 to 94.85, indicating possible outliers. PhysicalHealth and MentalHealth both have medians of 0 and maximums of 30, showing that many respondents reported no unhealthy days, while some experienced poor health for the entire month. SleepTime averages 7.08 hours, but the maximum of 24 hours suggests potential outliers. Overall, the data is clean but may still require outlier handling.

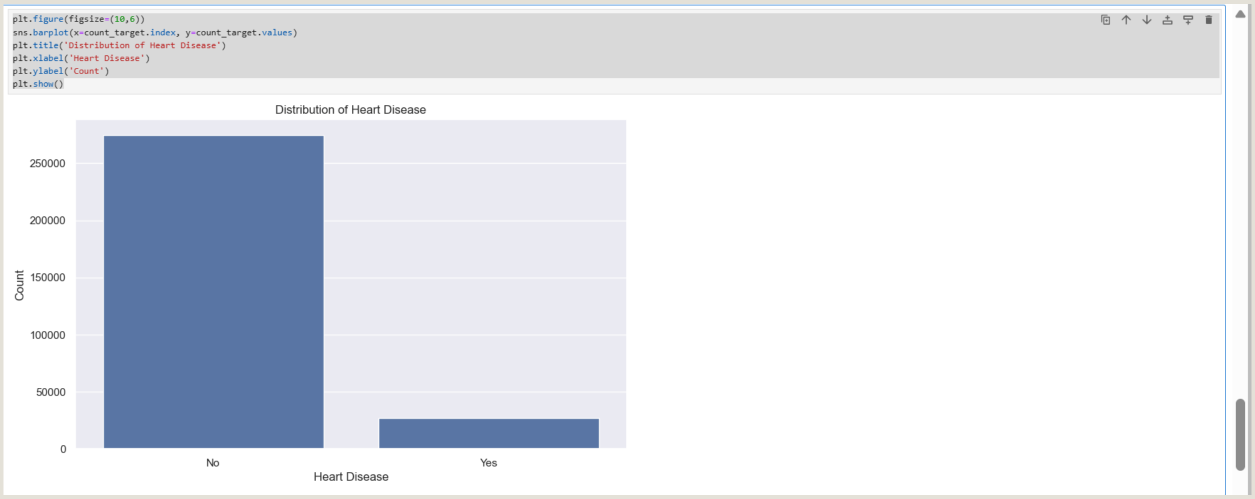
STEP 5

Data Exploration



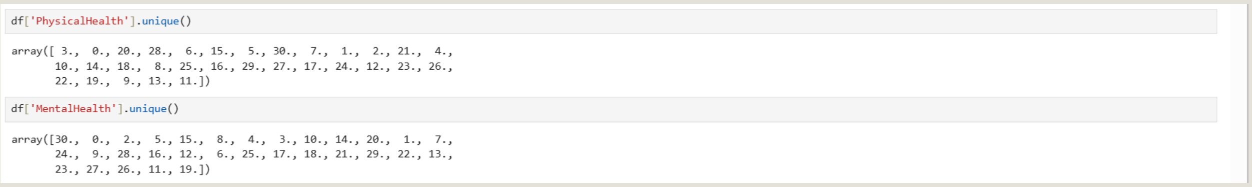
```
count_target = df['HeartDisease'].value_counts()
count_target
```

```
HeartDisease
No     274456
Yes     27261
Name: count, dtype: int64
```



```
plt.figure(figsize=(10,6))
sns.barplot(x=count_target.index, y=count_target.values)
plt.title('Distribution of Heart Disease')
plt.xlabel('Heart Disease')
plt.ylabel('Count')
plt.show()
```

I used count_target = df['HeartDisease'].value_counts() to count the occurrences of each category in the HeartDisease column, revealing 274,456 individuals without heart disease and 27,261 with, indicating a 9% positive case rate and class imbalance.



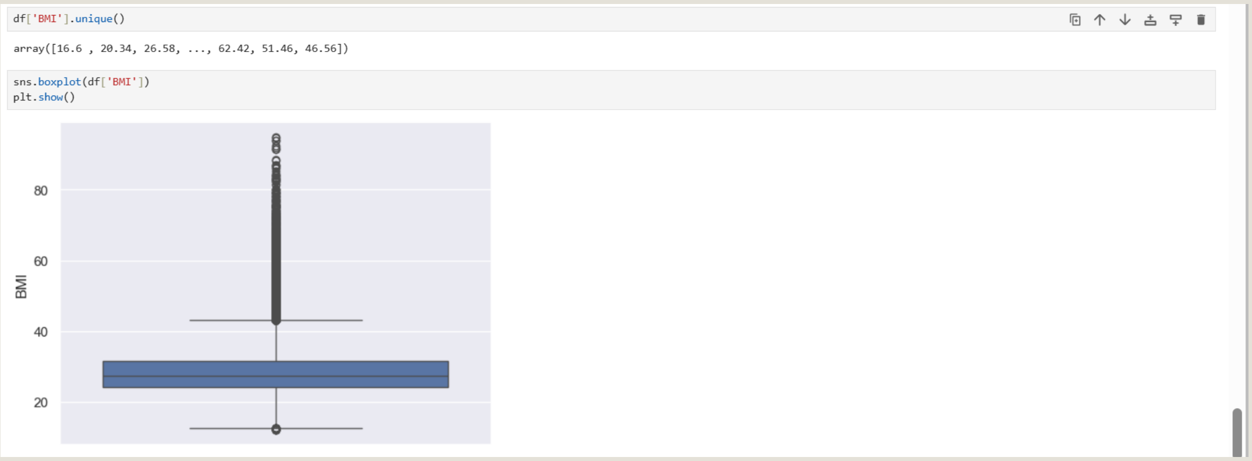
```
df['PhysicalHealth'].unique()
```

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

```
df['MentalHealth'].unique()
```

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

I used df['PhysicalHealth'].unique() and df['MentalHealth'].unique() to confirm that both columns contain values from 0 to 30, representing the number of unhealthy days in the past month and showing valid, bounded data.

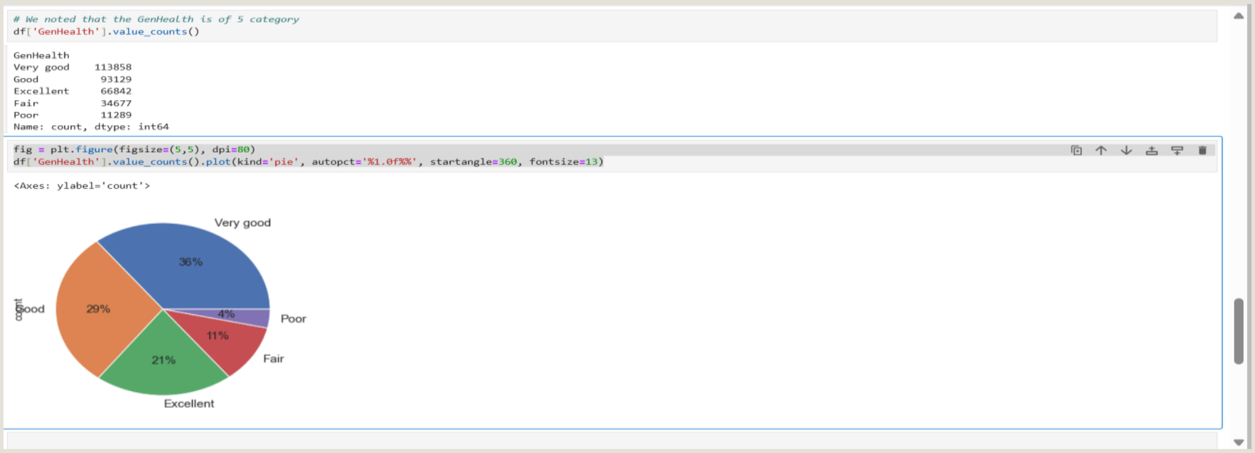


```
df['BMI'].unique()
```

```
array([16.6 , 20.34, 26.58, ..., 62.42, 51.46, 46.56])
```

```
sns.boxplot(df['BMI'])
plt.show()
```

I used df['BMI'].unique() to check the range of BMI values and sns.boxplot(df['BMI']) to visualize their distribution, which revealed the presence of outliers.



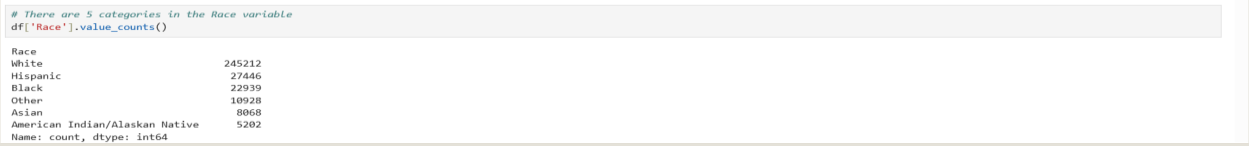
```
# We noted that the GenHealth is of 5 category
df['GenHealth'].value_counts()
```

```
GenHealth
Very good    113858
Good          93129
Excellent     66842
Fair          34677
Poor          11289
Name: count, dtype: int64
```

```
fig = plt.figure(figsize=(5,5), dpi=80)
df['GenHealth'].value_counts().plot(kind='pie', autopct='%1.0f%%', startangle=360, fontsize=13)
```

```
<Axes: ylabel='count'>
```

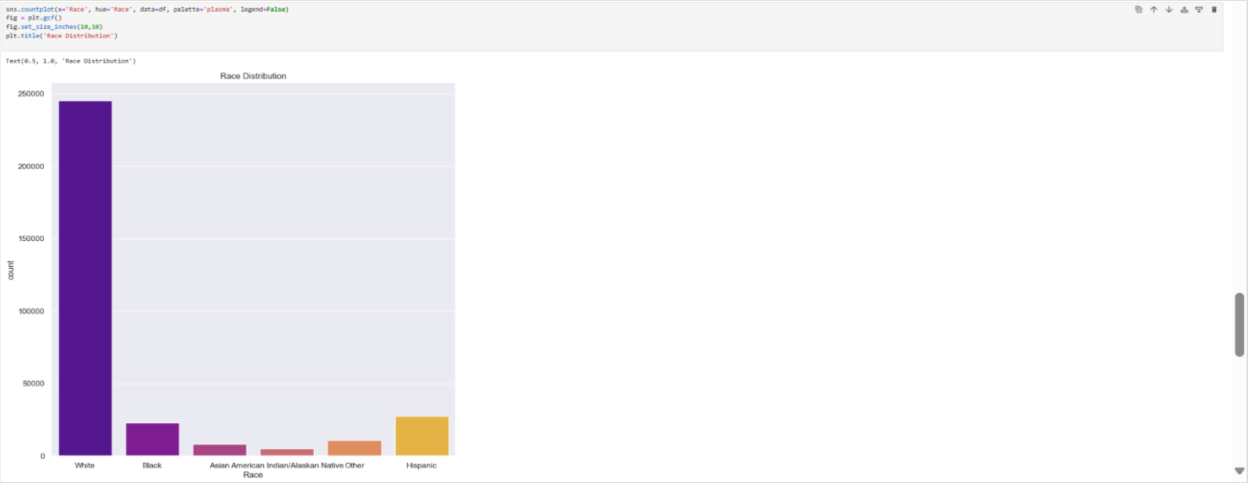
I used df['GenHealth'].value_counts() and a pie chart to observe that the GenHealth column has five categories, with "Very good" being the most common. The pie chart visually highlights the distribution, showing that most respondents rate their general health as "Very good," "Good," or "Excellent."



```
# There are 5 categories in the Race variable
df['Race'].value_counts()
```

```
Race
White                             245212
Hispanic                           27446
Black                              22939
Other                              10928
Asian                               8068
American Indian/Alaskan Native      5202
Name: count, dtype: int64
```

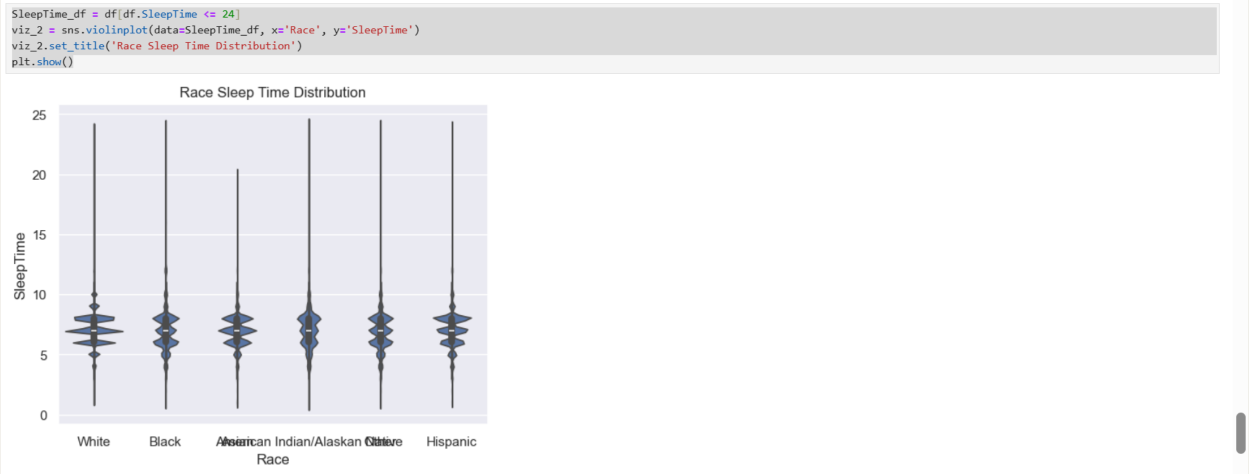
I used df['Race'].value_counts() to examine the distribution of the Race variable and identified six categories, with "White" being the most frequent. This analysis helps understand the demographic composition of the dataset.



```
sns.countplot(x='Race', hue='Race', data=df, palette='plasma', legend=False)
fig = plt.gcf()
fig.set_size_inches(10,10)
plt.title('Race Distribution')
```

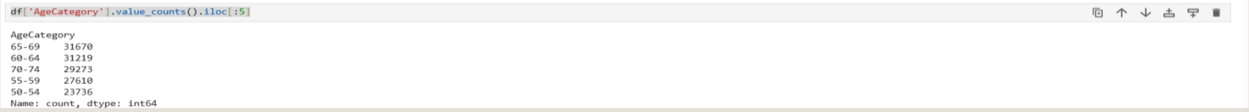
```
Text(0.5, 1.0, 'Race Distribution')
```

The bar chart shows a racial distribution dominated by the White category, which far exceeds all others in count. Black, Asian, and American Indian/Alaskan Native groups have notably lower counts, with the latter being the smallest. 'Other' and Hispanic categories have slightly higher counts than American Indian/Alaskan Native but remain relatively low. The overall distribution is heavily skewed toward the White population.



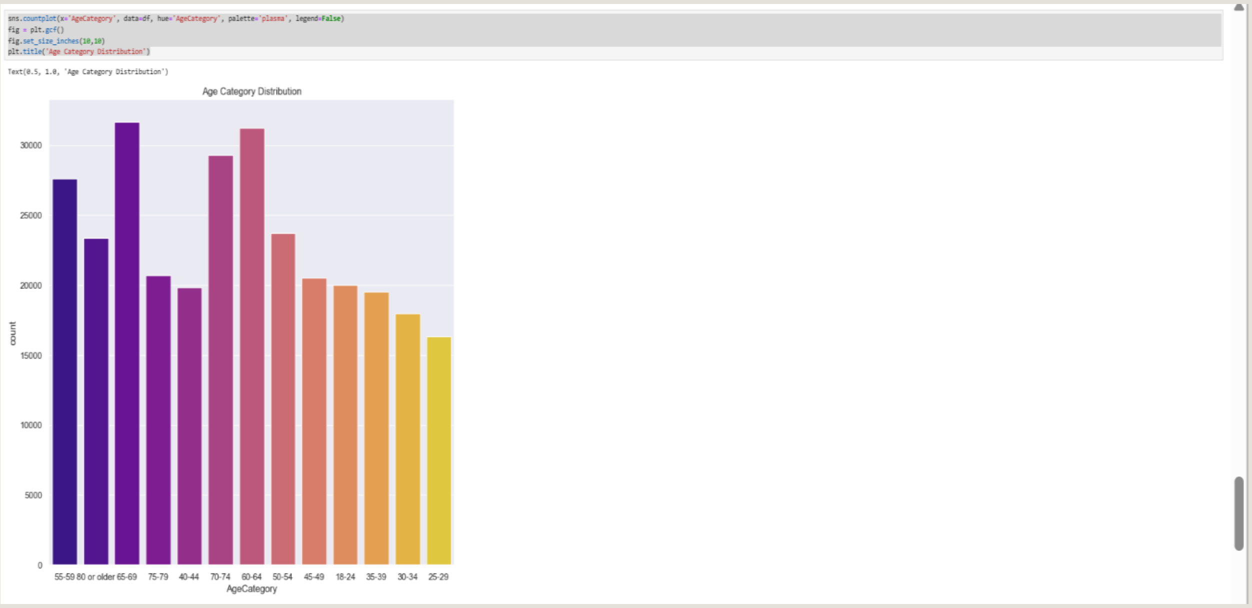
```
SleepTime_df = df[df.SleepTime <= 24]
viz_2 = sns.violinplot(data=SleepTime_df, x='Race', y='SleepTime')
viz_2.set_title('Race Sleep Time Distribution')
plt.show()
```

This analysis used a violin plot to compare self-reported sleep durations across racial groups. Despite filtering for sleep times of 24 hours or less. White respondents had the widest spread, while other groups showed more concentrated ranges. The visualization suggests differences in sleep behavior that may relate to broader social or health factors.



```
df['AgeCategory'].value_counts().iloc[:5]
```

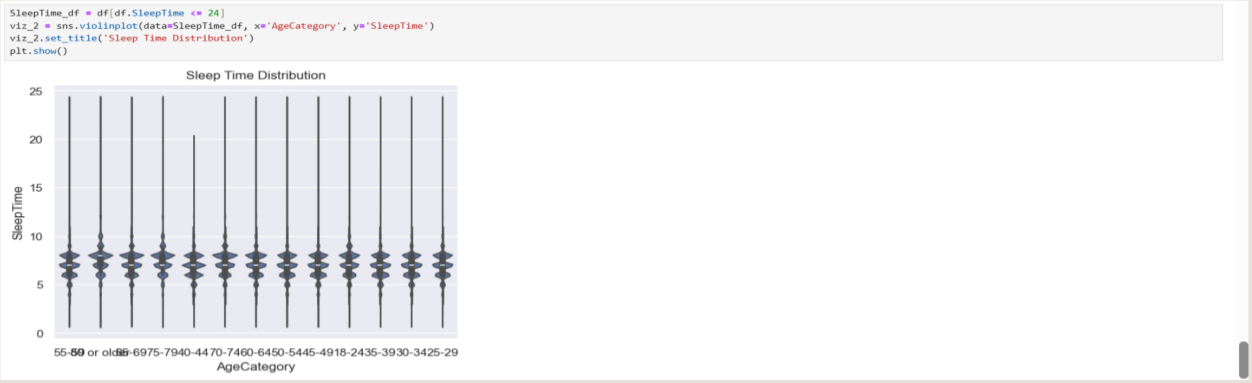
```
AgeCategory
65-69    31670
60-64    31219
70-74    29273
55-59    27610
50-54    23736
Name: count, dtype: int64
```



```
sns.countplot(x='AgeCategory', data=df, hue='AgeCategory', palette='plasma', legend=False)
fig = plt.gcf()
fig.set_size_inches(10,10)
plt.title('Age Category Distribution')
```

```
Text(0.5, 1.0, 'Age Category Distribution')
```

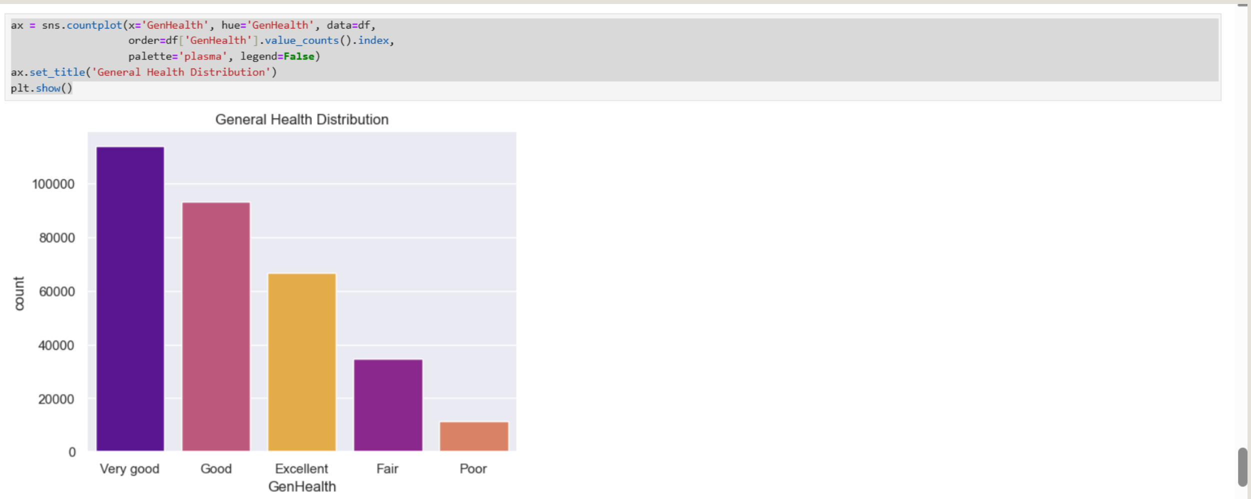
I used df['AgeCategory'].value_counts().iloc[:5] to identify the top five age categories, with 65-69 being the most common. I then visualized the age distribution with sns.countplot(x='AgeCategory', data=df, hue='AgeCategory', palette='plasma', legend=False), adjusting the figure size to 10x10 inches and adding a title for clarity.



```
SleepTime_df = df[df.SleepTime <= 24]
viz_2 = sns.violinplot(data=SleepTime_df, x='AgeCategory', y='SleepTime')
viz_2.set_title('Sleep Time Distribution')
plt.show()
```

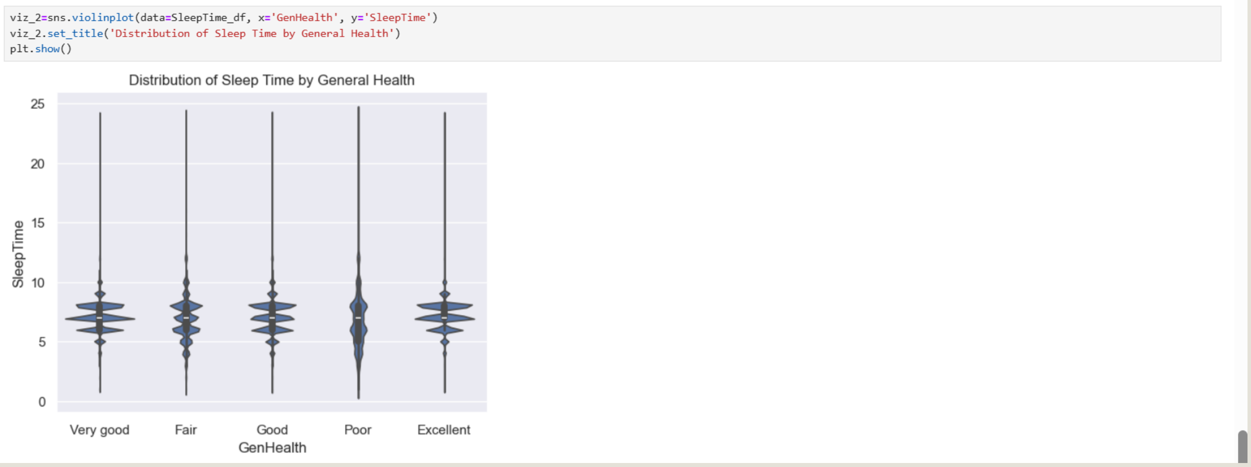
Sleep Time Distribution
SleepTime
0
5
10
15
20
25
55-5980 or old65-6975-7940-4470-7460-6450-5445-4918-2435-3930-3425-29
AgeCategory

I filtered the dataset to include only valid sleep times (SleepTime <= 24) and visualized the distribution of sleep times across age categories using a violin plot with the title "Sleep Time Distribution."



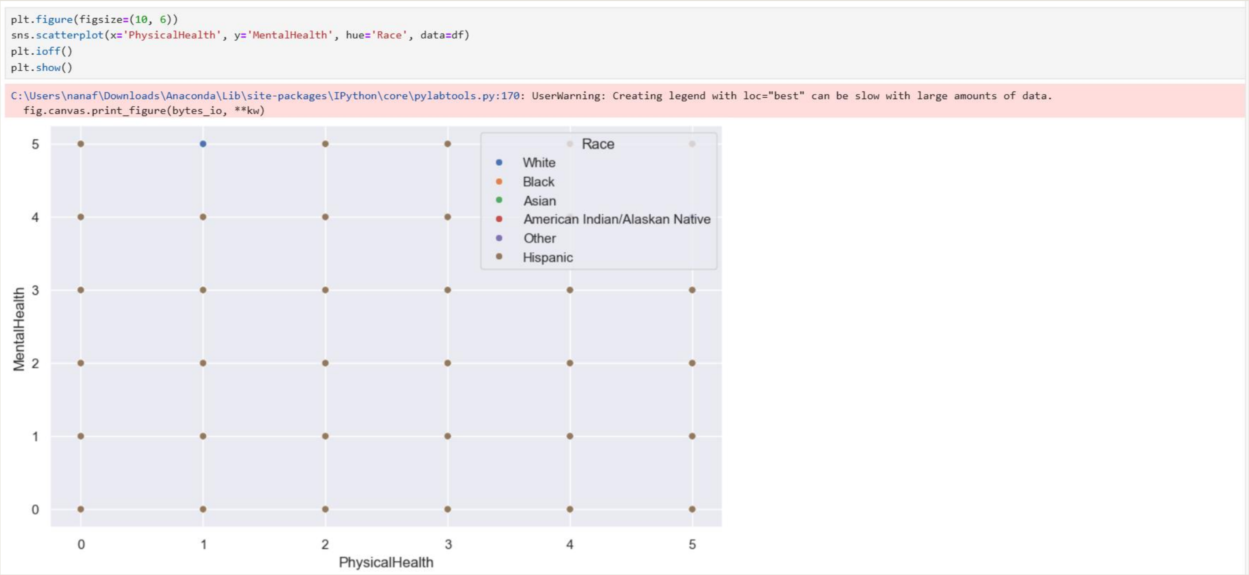
```
ax = sns.countplot(x='GenHealth', hue='GenHealth', data=df,
                   order=df['GenHealth'].value_counts().index,
                   palette='plasma', legend=False)
ax.set_title('General Health Distribution')
plt.show()
```

This analysis visualizes self-reported health from the GenHealth column using a Seaborn countplot with a "plasma" palette. "Very good" and "Good" were the most common responses, while "Poor" was least frequent, indicating overall positive health perception.



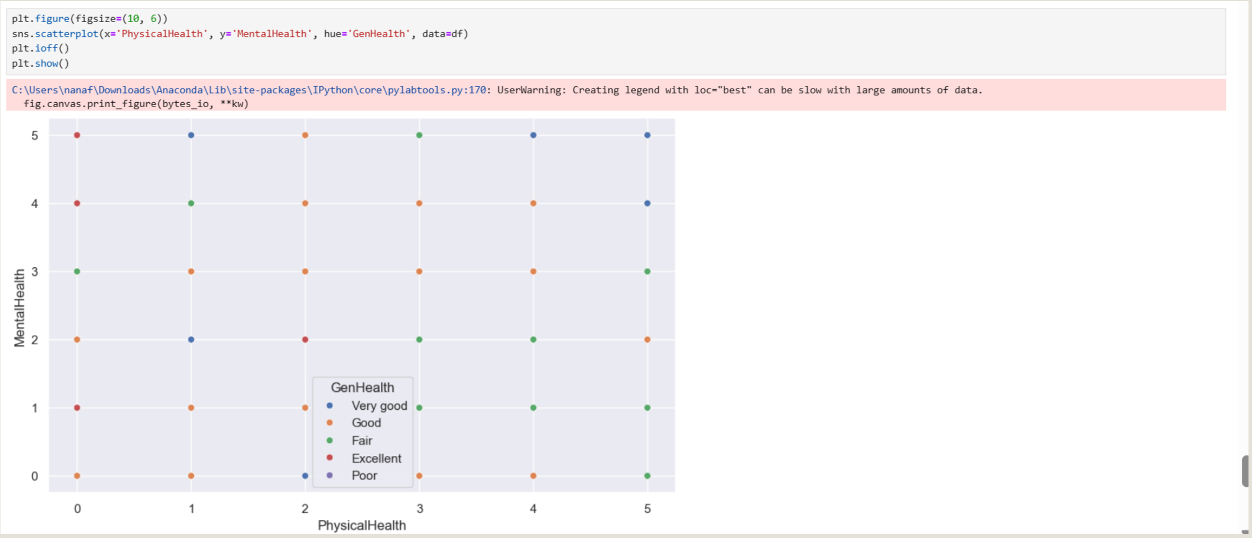
```
viz_2=sns.violinplot(data=SleepTime_df, x='GenHealth', y='SleepTime')
viz_2.set_title('Distribution of Sleep Time by General Health')
plt.show()
```

This analysis examines the link between general health and sleep duration using a violin plot. The plot shows sleep durations up to 24 hours, indicating filtering issues. People in very good health had the highest median sleep time (~20 hours), while those in fair/poor health had lower medians (~5–10 hours).



```
plt.figure(figsize=(10, 6))
sns.scatterplot(x='PhysicalHealth', y='MentalHealth', hue='Race', data=df)
plt.ioff()
plt.show()
```

```
C:\Users\nanaf\Downloads\Anaconda\Lib\site-packages\IPython\core\pylabtools.py:170: UserWarning: Creating legend with loc="best" can be slow with large amounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```

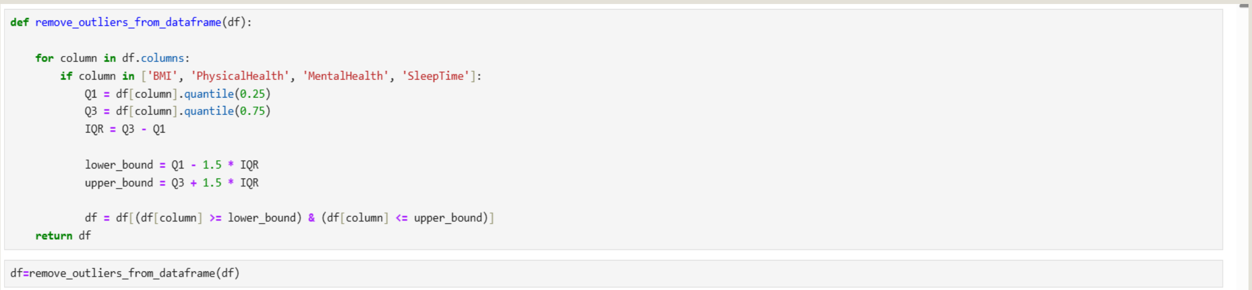


```
plt.figure(figsize=(10, 6))
sns.scatterplot(x='PhysicalHealth', y='MentalHealth', hue='GenHealth', data=df)
plt.ioff()
plt.show()
```

```
C:\Users\nanaf\Downloads\Anaconda\Lib\site-packages\IPython\core\pylabtools.py:170: UserWarning: Creating legend with loc="best" can be slow with large amounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```

STEP 6

Handling Outliers



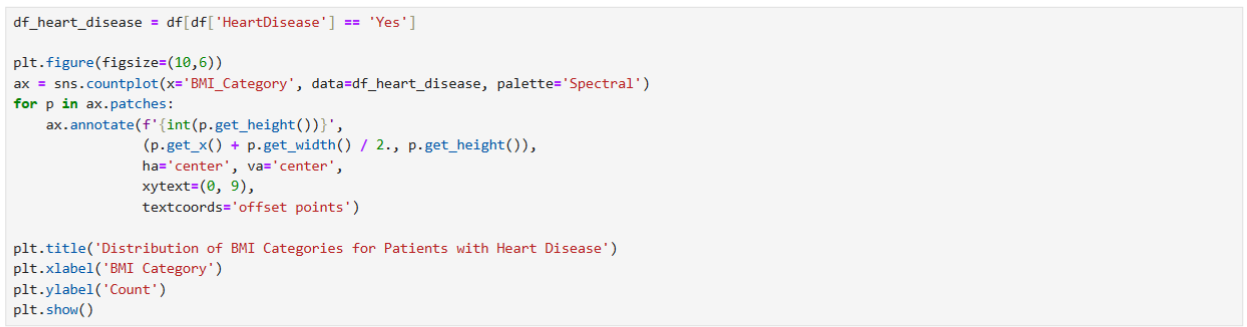
```
def remove_outliers_from_dataframe(df):

    for column in df.columns:
        if column in ['BMI', 'PhysicalHealth', 'MentalHealth', 'SleepTime']:
            Q1 = df[column].quantile(0.25)
            Q3 = df[column].quantile(0.75)
            IQR = Q3 - Q1

            lower_bound = Q1 - 1.5 * IQR
            upper_bound = Q3 + 1.5 * IQR

            df = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]
    return df
```

```
df=remove_outliers_from_dataframe(df)
```

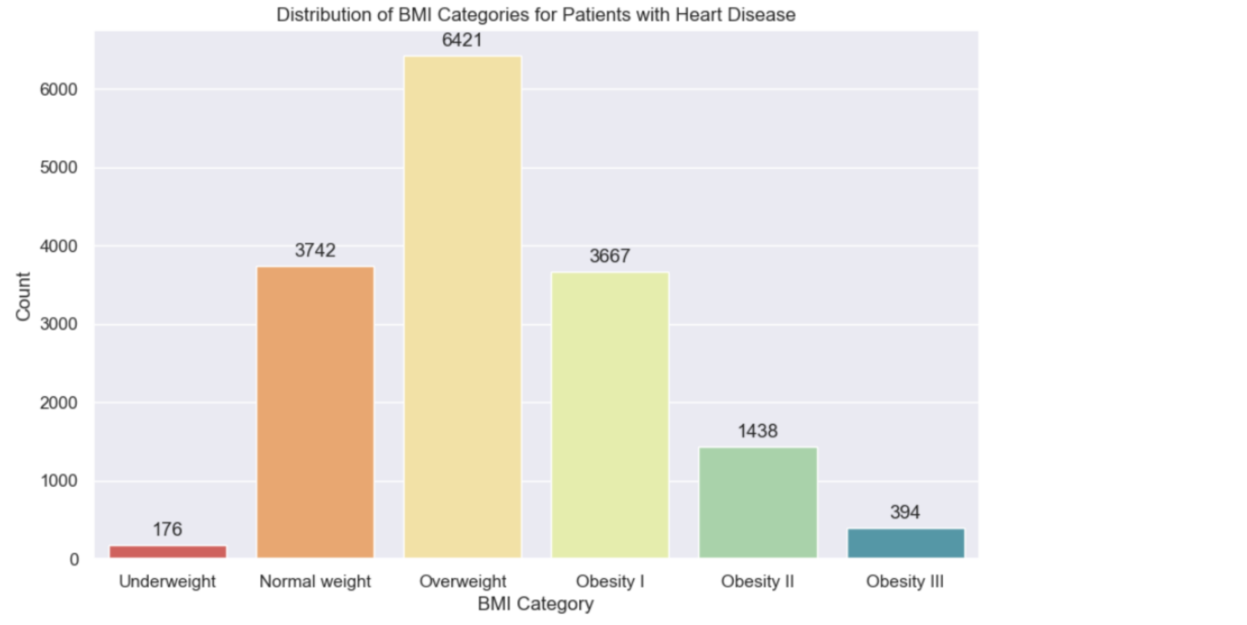
The dataset was cleaned by removing outliers from BMI, PhysicalHealth, MentalHealth, and SleepTime using the IQR method. Values outside 1.5 times the interquartile range were excluded to improve data quality for modeling.

STEP 7

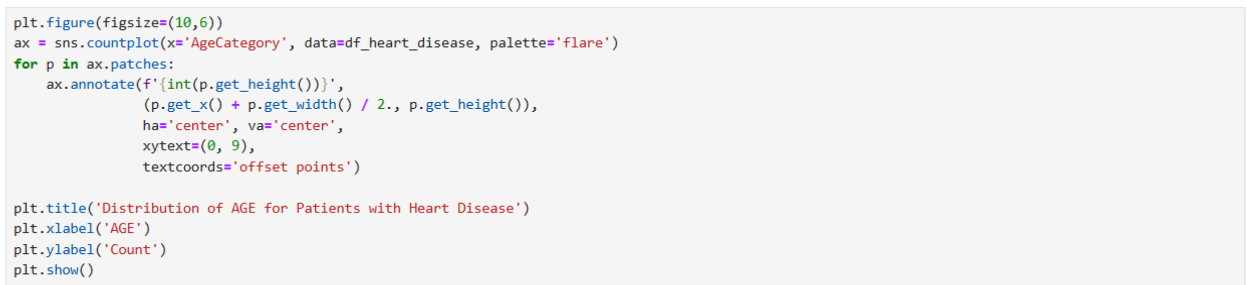
Data Visualization



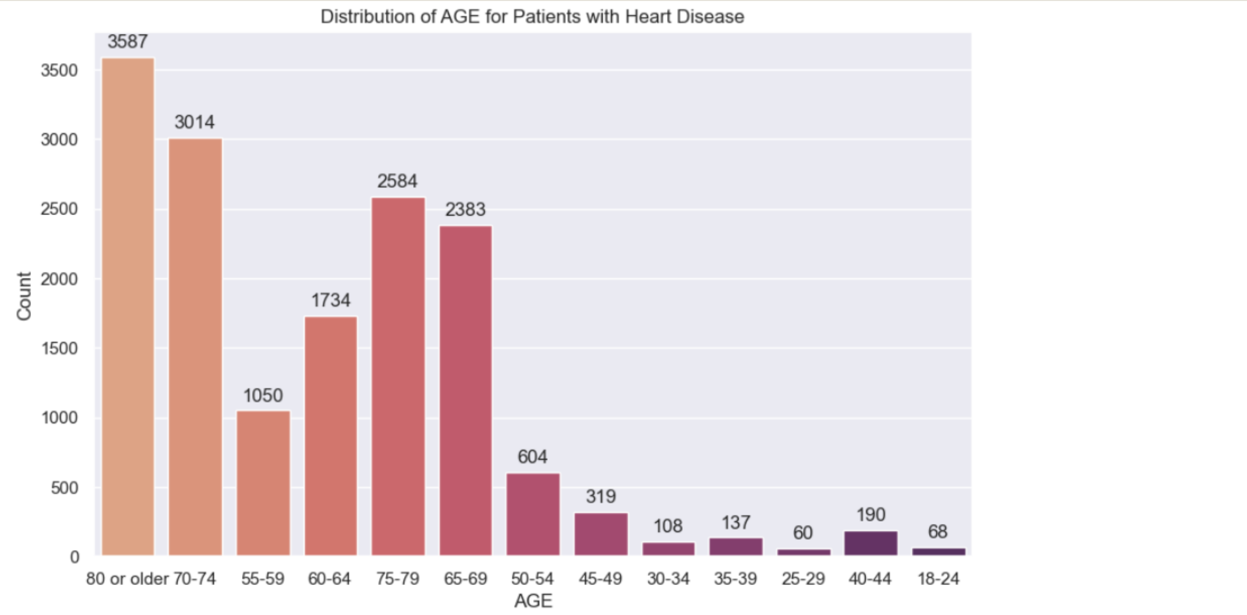
```
df_heart_disease = df[df['HeartDisease'] == 'Yes']

plt.figure(figsize=(10,6))
ax = sns.countplot(x='BMI_Category', data=df_heart_disease, palette='Spectral')
for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}',
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center',
                xytext=(0, 9),
                textcoords='offset points')

plt.title('Distribution of BMI Categories for Patients with Heart Disease')
plt.xlabel('BMI Category')
plt.ylabel('Count')
plt.show()
```



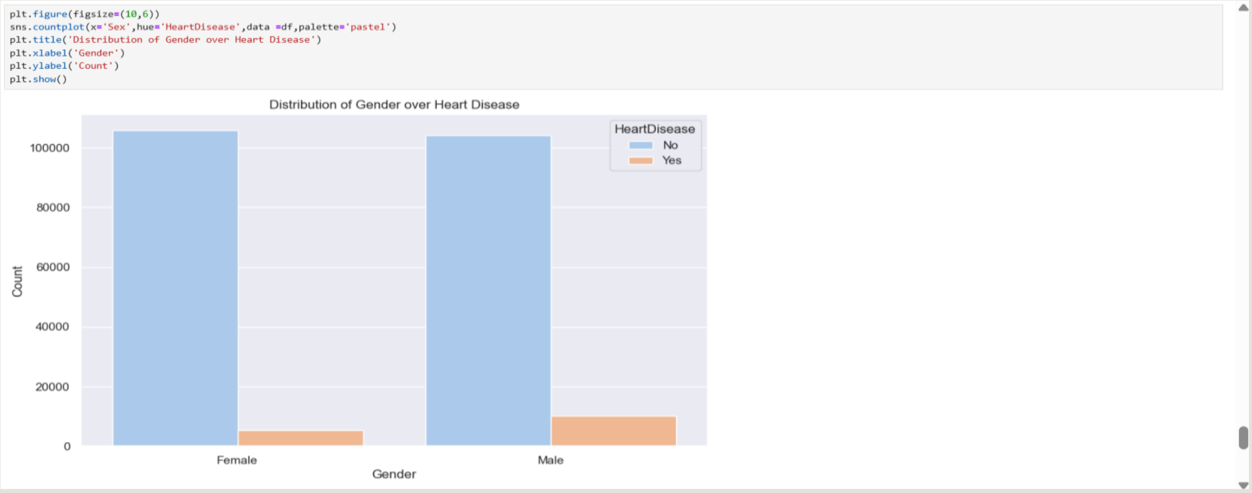
I examined the distribution of BMI categories among patients with heart disease. The analysis reveals that the majority fall into the "Overweight" category (6,407), followed closely by "Obesity I" (3,622) and "Normal weight" (3,710). Notably fewer patients are in the "Obesity II" (1,417), "Obesity III" (302), and "Underweight" (173) categories. This indicates a strong prevalence of heart disease among individuals with elevated BMI, especially those classified as overweight or moderately obese.



```
plt.figure(figsize=(10,6))
ax = sns.countplot(x='AgeCategory', data=df_heart_disease, palette='flare')
for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}',
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center',
                xytext=(0, 9),
                textcoords='offset points')

plt.title('Distribution of AGE for Patients with Heart Disease')
plt.xlabel('AGE')
plt.ylabel('Count')
plt.show()
```

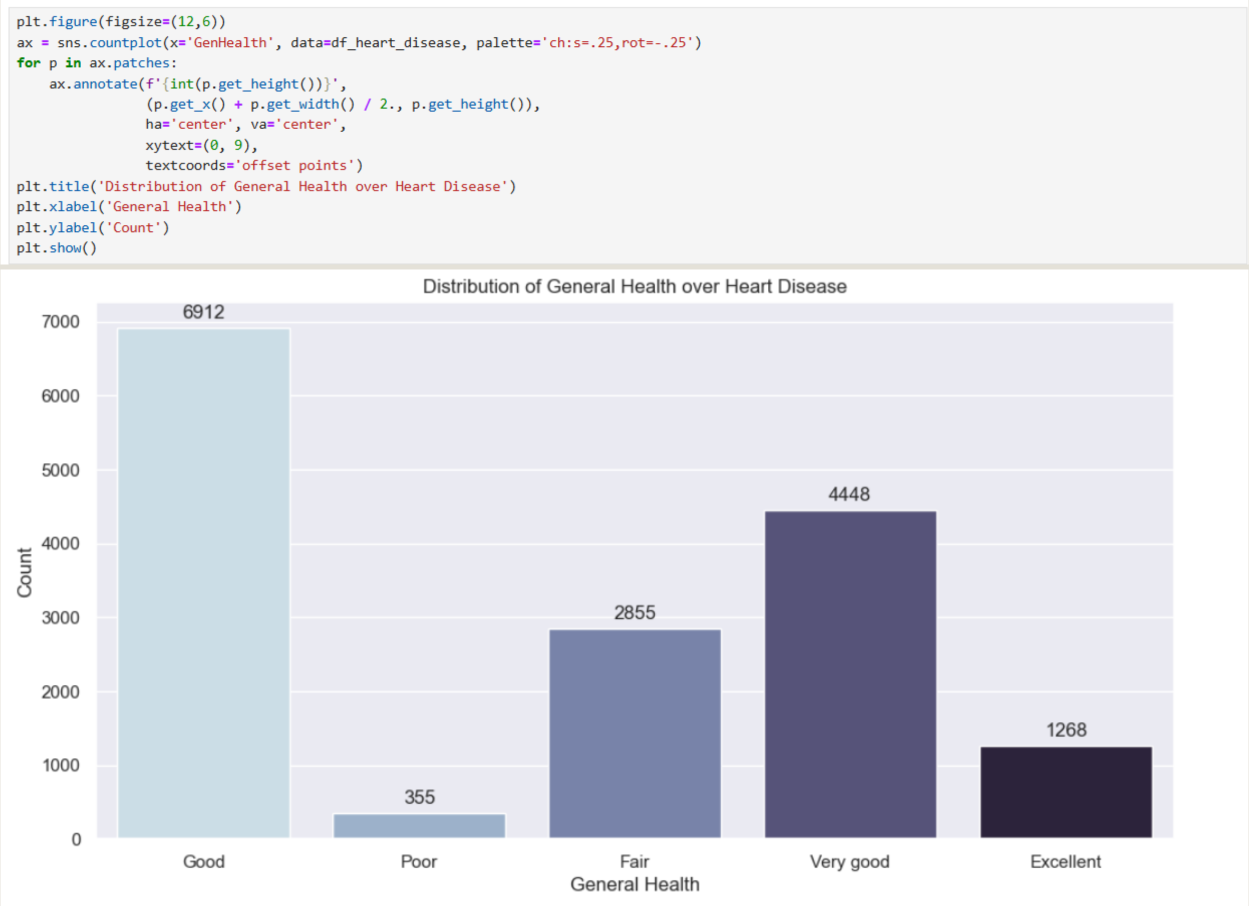


I analyzed the distribution of age among patients with heart disease using a count plot. The results show that heart disease is most common in older adults, with the highest number of cases in the "80 or older" group (3,576), followed by "70–74" (2,999) and "75–79" (2,582). The number of cases decreases steadily in younger age groups, with fewer than 200 cases in each group under 45 years. The lowest count appears in the "18–24" group (66). This confirms that the risk of heart disease increases significantly with age.



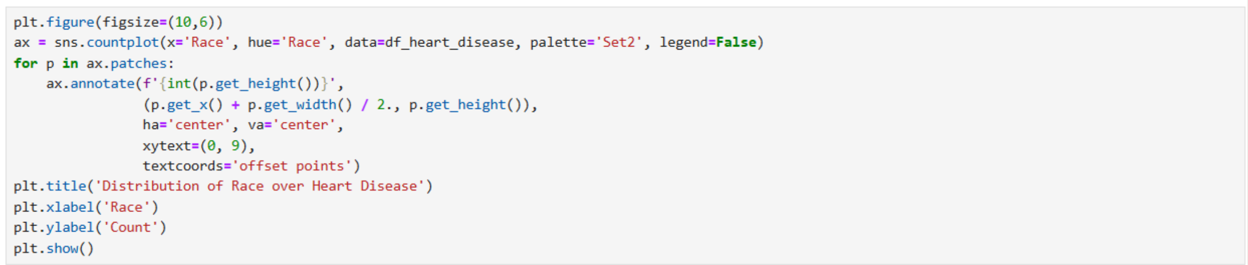
```
plt.figure(figsize=(10,6))
sns.countplot(x='Sex',hue='HeartDisease',data =df,palette='pastel')
plt.title('Distribution of Gender over Heart Disease')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.show()
```

I analyzed heart disease distribution by gender using a countplot. I filtered the dataset to compare males and females based on heart disease status, with gender on the x-axis and heart disease as the hue. The plot showed that around 100,000 females had no heart disease and about 5,000 did, while about 100,000 males had no heart disease and about 10,000 did. This indicates a higher number of cases in males.

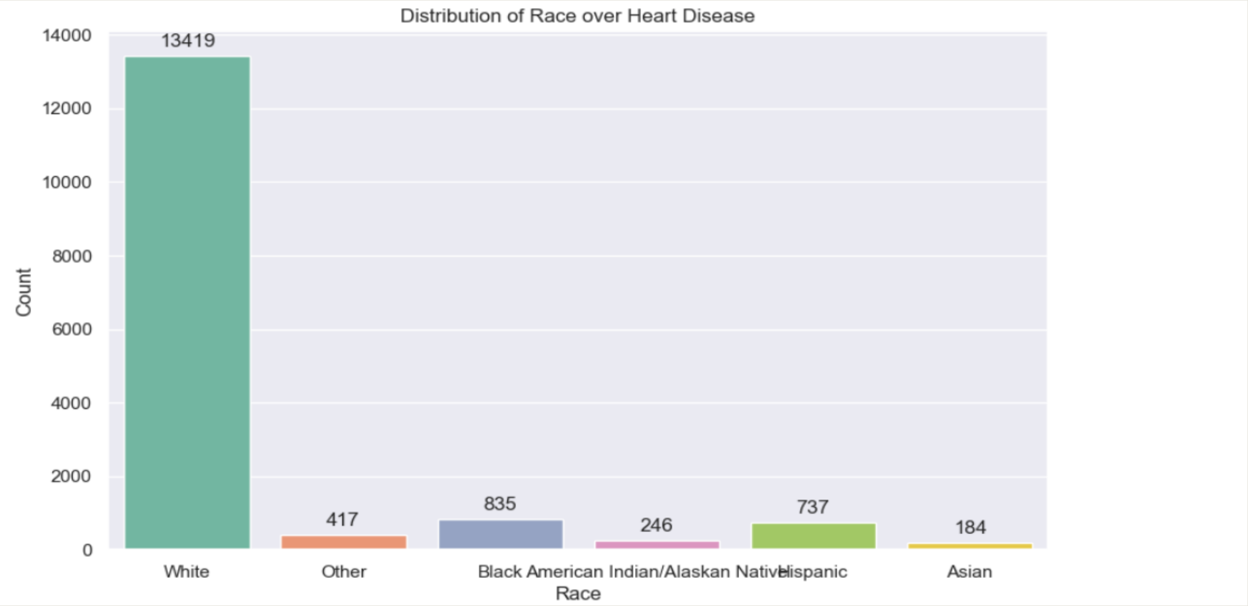


```
plt.figure(figsize=(12,6))
ax = sns.countplot(x='GenHealth', data=df_heart_disease, palette='ch:s=.25,rot=-.25')
for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}',
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center',
                xytext=(0, 9),
                textcoords='offset points')
plt.title('Distribution of General Health over Heart Disease')
plt.xlabel('General Health')
plt.ylabel('Count')
plt.show()
```

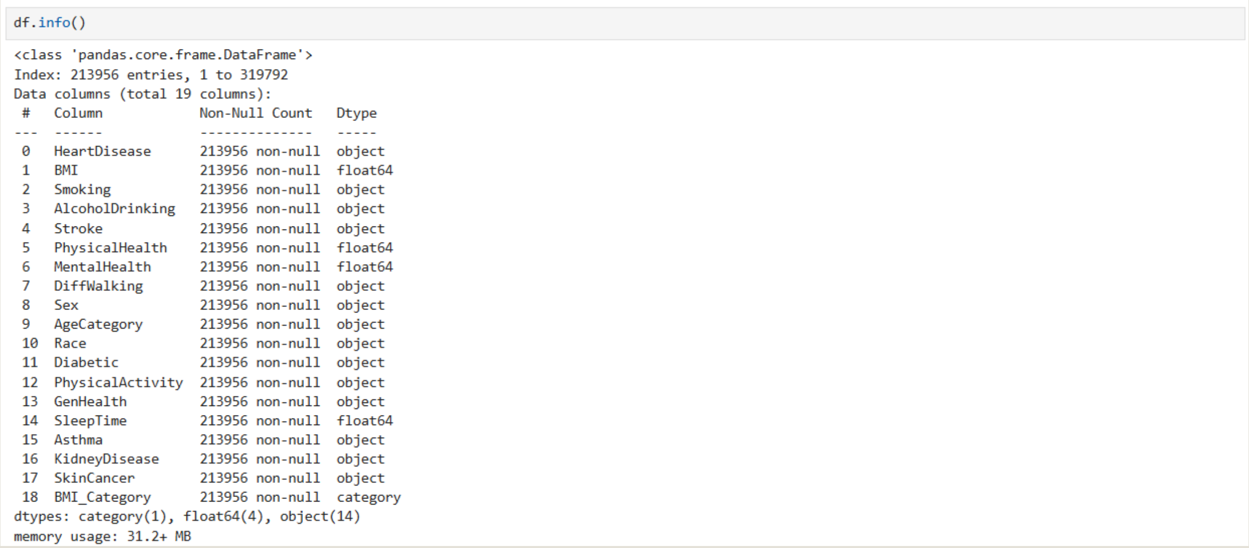
This analysis explores the distribution of heart disease cases across self-reported general health levels. The countplot shows that most individuals with heart disease reported "Good" or "Very good" health, despite these being less indicative of actual health status. This suggests the importance of regular medical screenings beyond self-assessment.



```python
plt.figure(figsize=(10,6))
ax = sns.countplot(x='Race', hue='Race', data=df_heart_disease, palette='Set2', legend=False)
for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}',
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center',
                xytext=(0, 9),
                textcoords='offset points')
plt.title('Distribution of Race over Heart Disease')
plt.xlabel('Race')
plt.ylabel('Count')
plt.show()
```

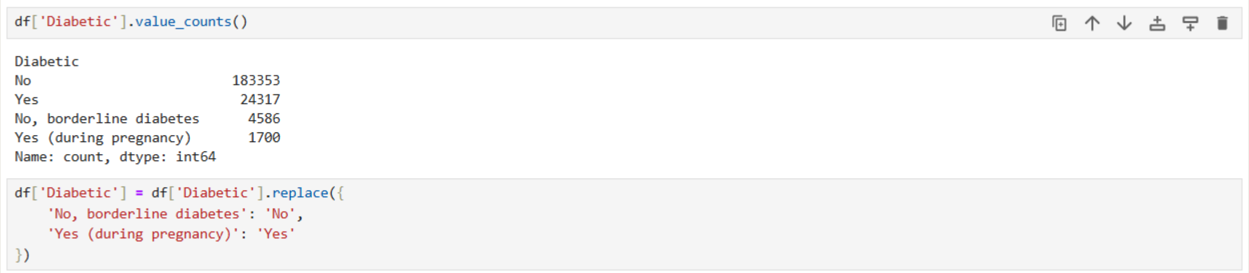


I analyzed the racial distribution in the heart disease dataset using a countplot. Most participants were White (13,276), with much smaller counts for other groups—Black (823), Hispanic (710), Other (710), American Indian/Alaskan Native (241), and Asian (180).



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 213956 entries, 1 to 319792
Data columns (total 19 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   HeartDisease      213956 non-null  object
 1   BMI               213956 non-null  float64
 2   Smoking           213956 non-null  object
 3   AlcoholDrinking   213956 non-null  object
 4   Stroke            213956 non-null  object
 5   PhysicalHealth    213956 non-null  float64
 6   MentalHealth      213956 non-null  float64
 7   DiffWalking       213956 non-null  object
 8   Sex               213956 non-null  object
 9   AgeCategory       213956 non-null  object
 10  Race              213956 non-null  object
 11  Diabetic          213956 non-null  object
 12  PhysicalActivity  213956 non-null  object
 13  GenHealth         213956 non-null  object
 14  SleepTime         213956 non-null  float64
 15  Asthma            213956 non-null  object
 16  KidneyDisease     213956 non-null  object
 17  SkinCancer        213956 non-null  object
 18  BMI_Category      213956 non-null  category
dtypes: category(1), float64(4), object(14)
memory usage: 31.2+ MB
```

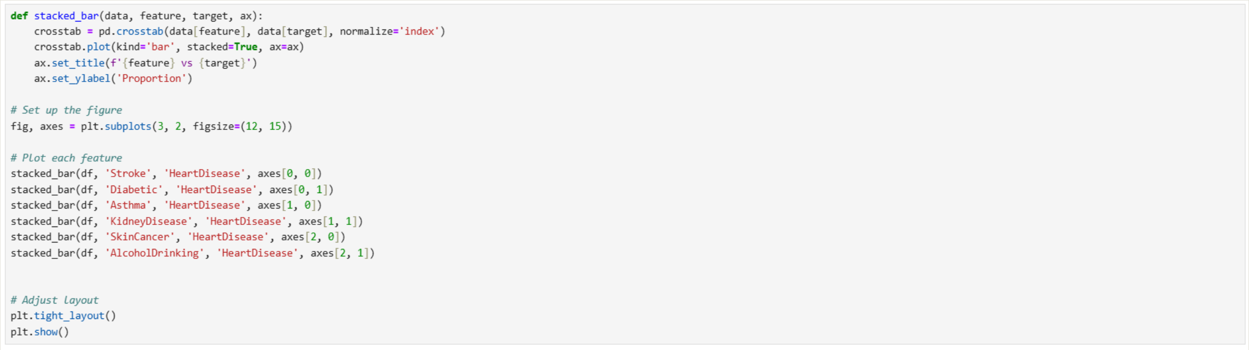


```
df['Diabetic'].value_counts()
```

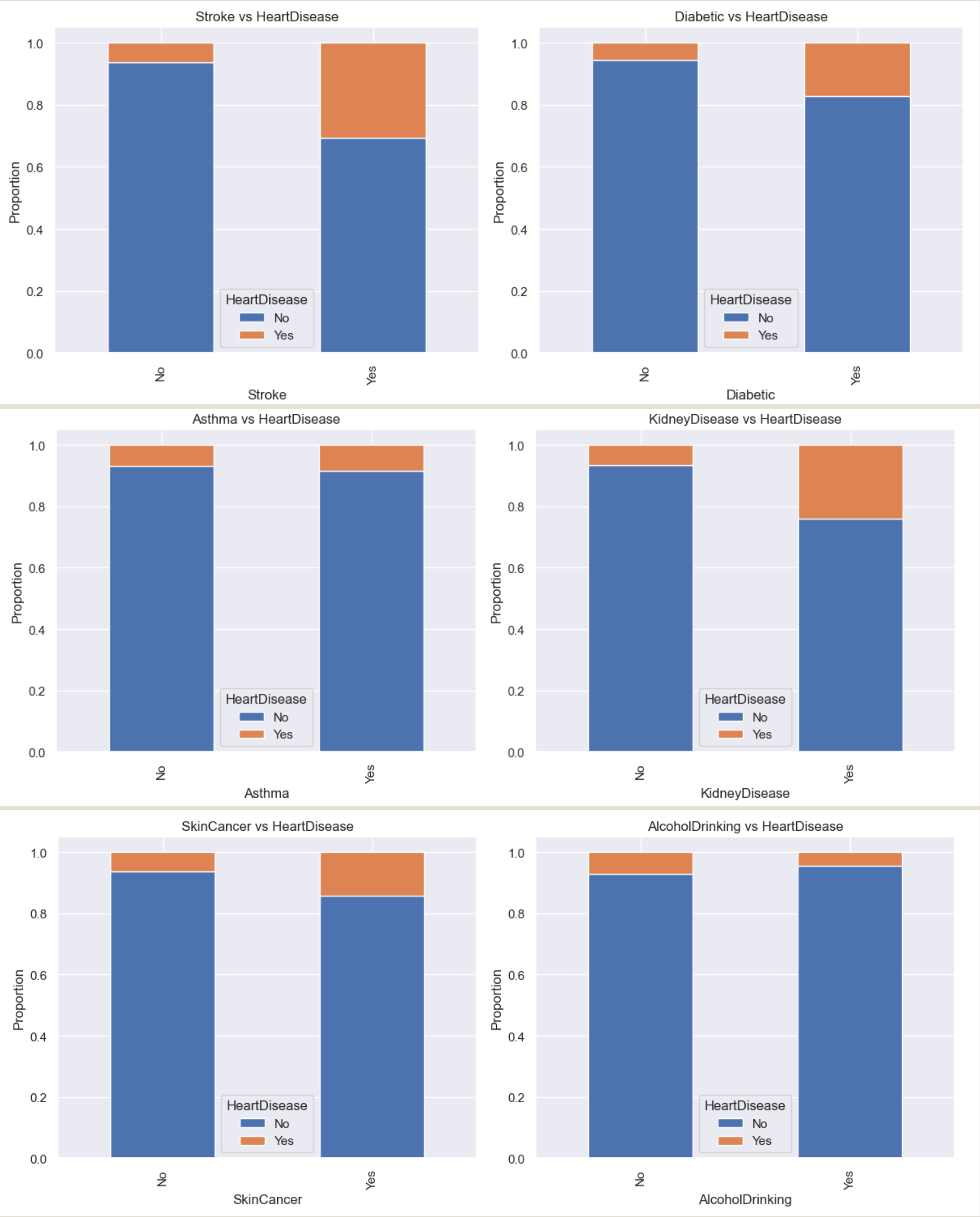
```
Diabetic
No                           183353
Yes                           24317
No, borderline diabetes        4586
Yes (during pregnancy)         1700
Name: count, dtype: int64
```

```
df['Diabetic'] = df['Diabetic'].replace({
    'No, borderline diabetes': 'No',
    'Yes (during pregnancy)': 'Yes'
})
```

I examined the distribution of responses in the Diabetic column and found four categories: "No", "Yes", "No, borderline diabetes", and "Yes (during pregnancy)". To simplify the data, I grouped similar responses by replacing "No, borderline diabetes" with "No" and "Yes (during pregnancy)" with "Yes". This created a cleaner binary classification of diabetic status, making the data easier to analyze.

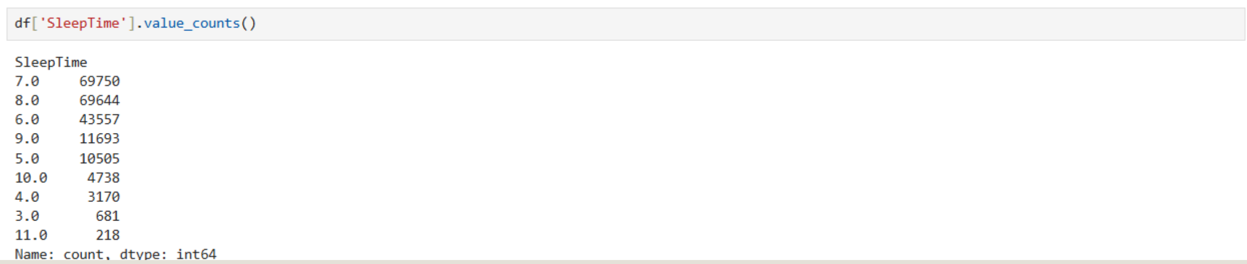


```
def stacked_bar(data, feature, target, ax):
    crosstab = pd.crosstab(data[feature], data[target], normalize='index')
    crosstab.plot(kind='bar', stacked=True, ax=ax)
    ax.set_title(f'{feature} vs {target}')
    ax.set_ylabel('Proportion')

# Set up the figure
fig, axes = plt.subplots(3, 2, figsize=(12, 15))

# Plot each feature
stacked_bar(df, 'Stroke', 'HeartDisease', axes[0, 0])
stacked_bar(df, 'Diabetic', 'HeartDisease', axes[0, 1])
stacked_bar(df, 'Asthma', 'HeartDisease', axes[1, 0])
stacked_bar(df, 'KidneyDisease', 'HeartDisease', axes[1, 1])
stacked_bar(df, 'SkinCancer', 'HeartDisease', axes[2, 0])
stacked_bar(df, 'AlcoholDrinking', 'HeartDisease', axes[2, 1])


# Adjust layout
plt.tight_layout()
plt.show()
```

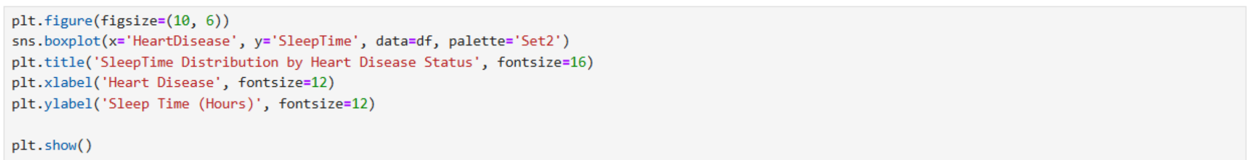


Stacked bar plots were created to visualize the relationship between several health conditions and heart disease. Features such as Stroke, Diabetic, Asthma, KidneyDisease, SkinCancer, and

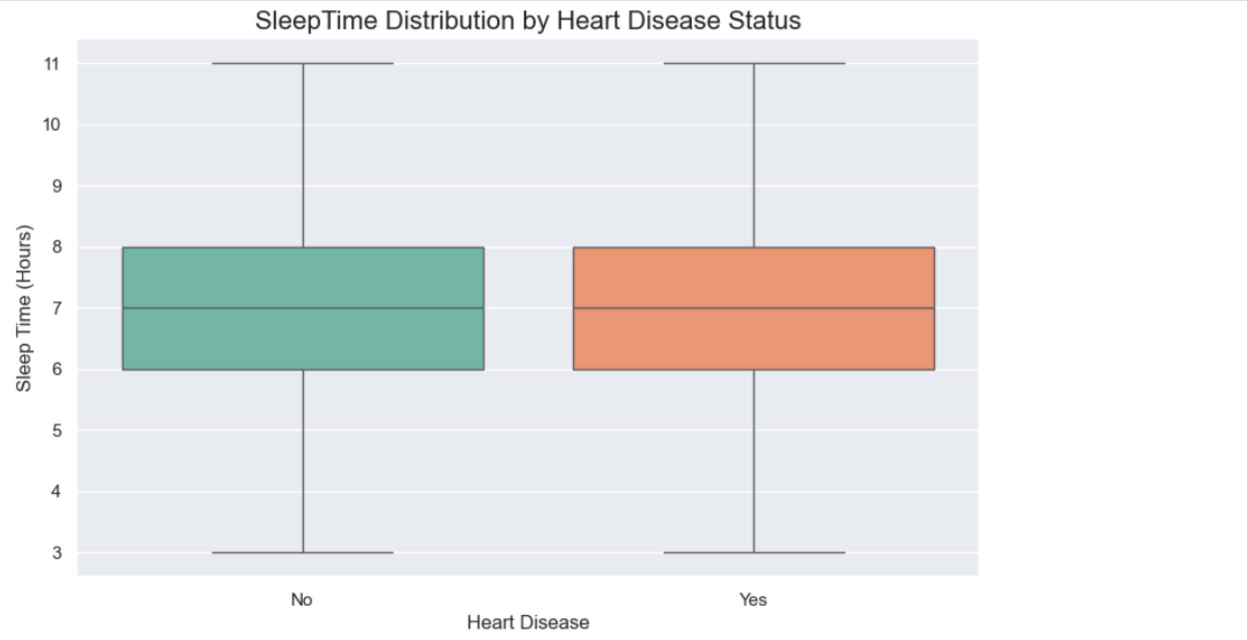
AlcoholDrinking were plotted against HeartDisease to observe proportional differences. This helped identify feature correlations useful for modeling.



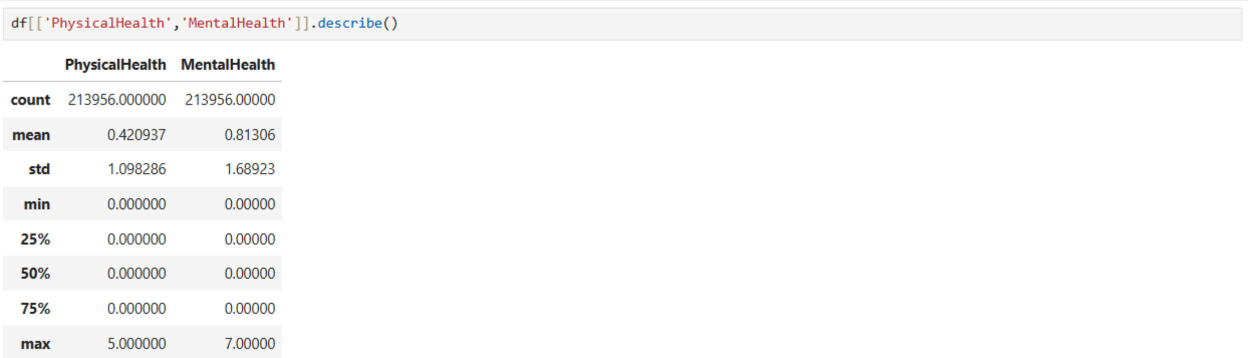
```
df['SleepTime'].value_counts()

SleepTime
7.0     69750
8.0     69644
6.0     43557
9.0     11693
5.0     10505
10.0     4738
4.0      3170
3.0       681
11.0      218
Name: count, dtype: int64
```



```
plt.figure(figsize=(10, 6))
sns.boxplot(x='HeartDisease', y='SleepTime', data=df, palette='Set2')
plt.title('SleepTime Distribution by Heart Disease Status', fontsize=16)
plt.xlabel('Heart Disease', fontsize=12)
plt.ylabel('Sleep Time (Hours)', fontsize=12)

plt.show()
```

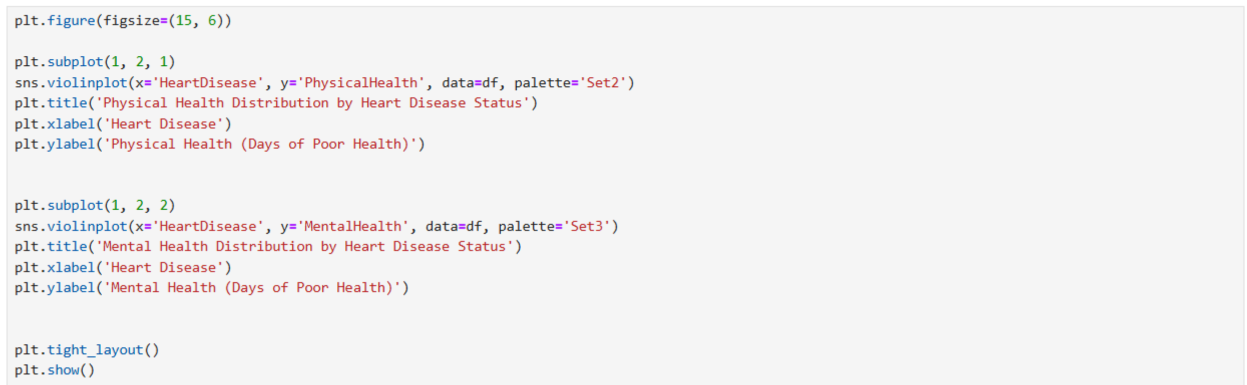


A boxplot was used to visualize the distribution of Sleep Time across Heart Disease statuses. This plot helped assess differences in sleep patterns between individuals with and without heart disease, revealing potential insights into the impact of sleep duration on heart health.

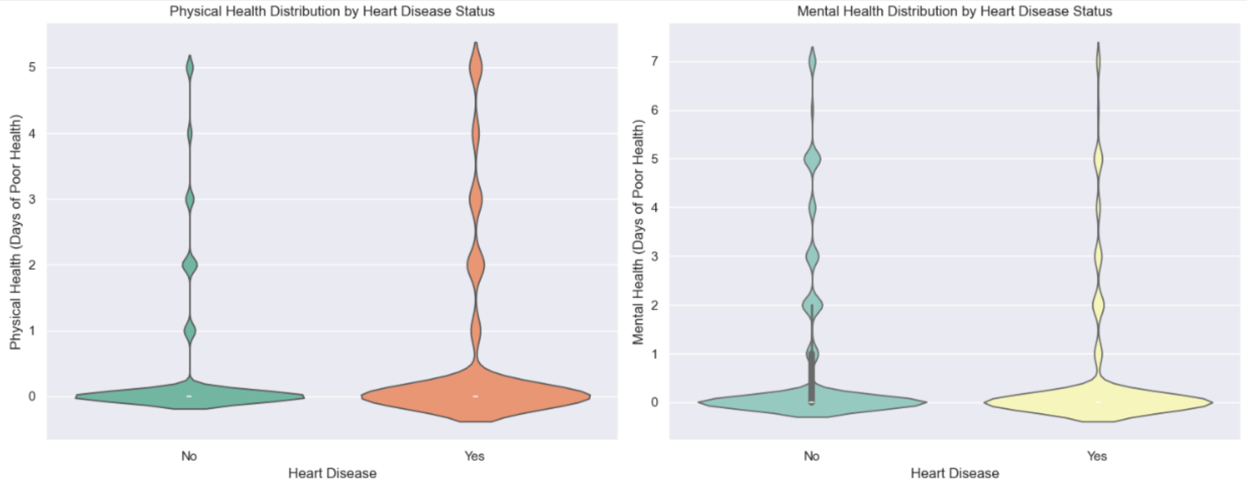


```
df[['PhysicalHealth','MentalHealth']].describe()
```

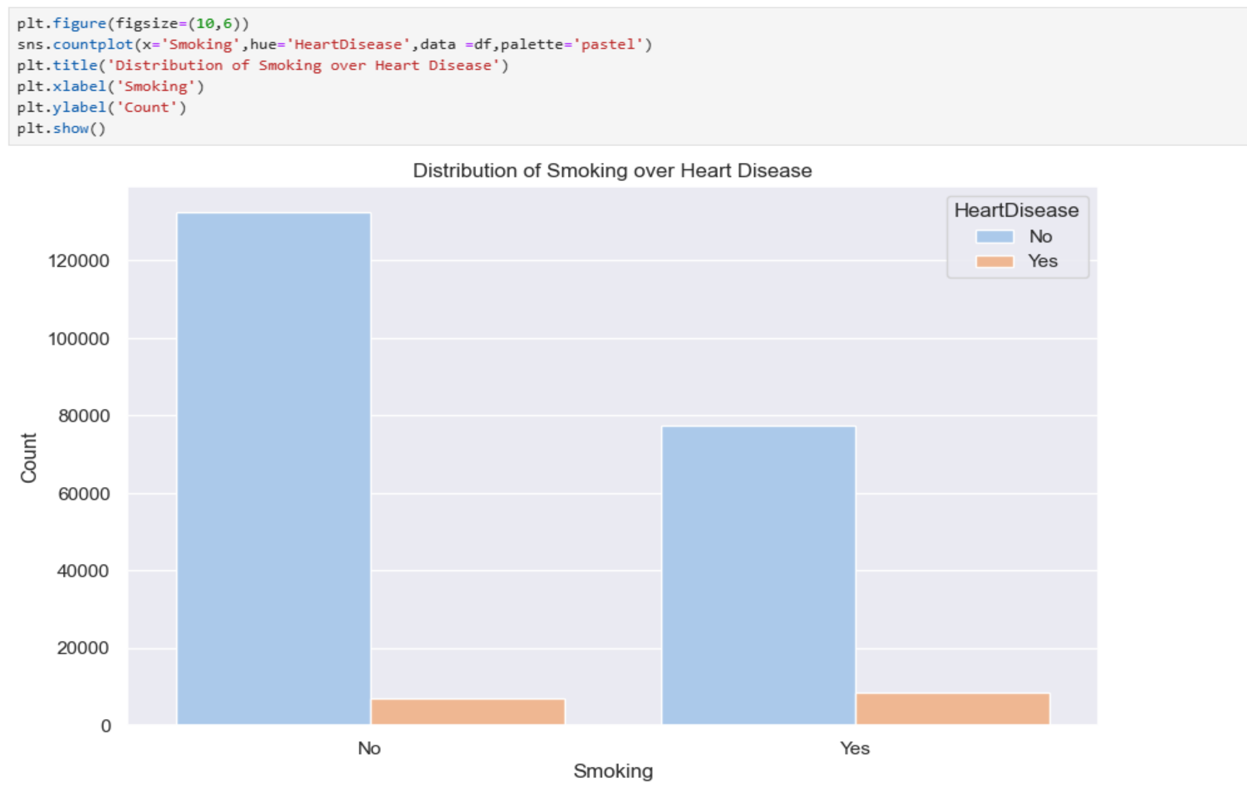
	PhysicalHealth	MentalHealth
count	213956.000000	213956.00000
mean	0.420937	0.81306
std	1.098286	1.68923
min	0.000000	0.00000
25%	0.000000	0.00000
50%	0.000000	0.00000
75%	0.000000	0.00000
max	5.000000	7.00000



```
plt.figure(figsize=(15, 6))

plt.subplot(1, 2, 1)
sns.violinplot(x='HeartDisease', y='PhysicalHealth', data=df, palette='Set2')
plt.title('Physical Health Distribution by Heart Disease Status')
plt.xlabel('Heart Disease')
plt.ylabel('Physical Health (Days of Poor Health)')


plt.subplot(1, 2, 2)
sns.violinplot(x='HeartDisease', y='MentalHealth', data=df, palette='Set3')
plt.title('Mental Health Distribution by Heart Disease Status')
plt.xlabel('Heart Disease')
plt.ylabel('Mental Health (Days of Poor Health)')


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

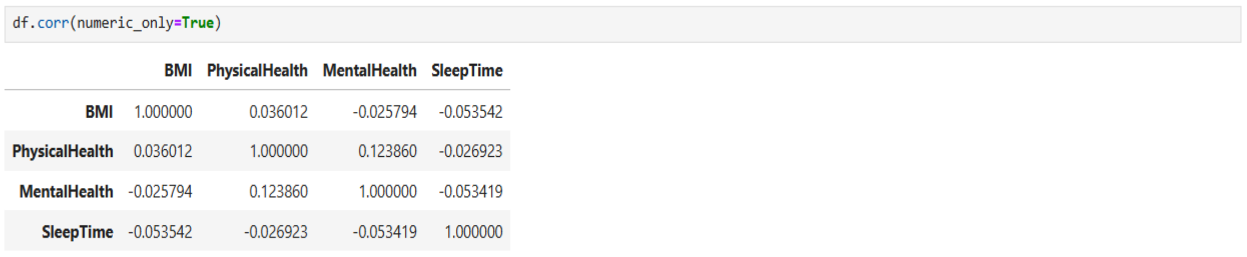


Violin plots showed that individuals with heart disease experienced more days of poor physical and mental health. This indicates a possible link between heart disease and reduced well-being.



```
plt.figure(figsize=(10,6))
sns.countplot(x='Smoking',hue='HeartDisease',data =df,palette='pastel')
plt.title('Distribution of Smoking over Heart Disease')
plt.xlabel('Smoking')
plt.ylabel('Count')
plt.show()
```

A count plot was used to compare smoking status among individuals with and without heart disease. The plot showed the number of smokers and non-smokers in each group. A higher number of smokers among those with heart disease suggests a possible link.

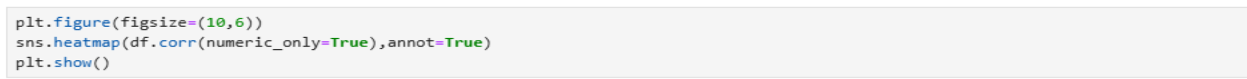


```
df.corr(numeric_only=True)
```

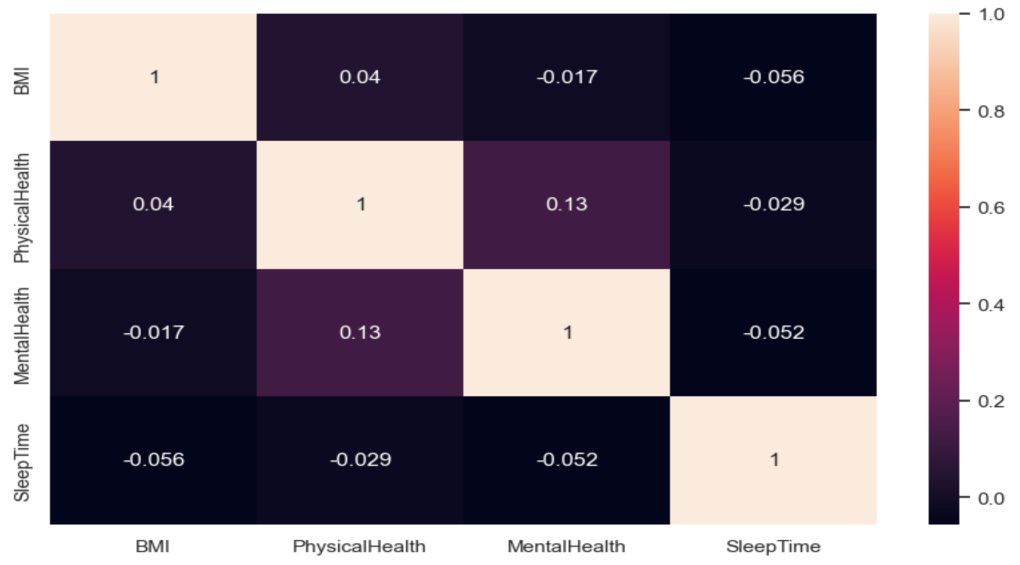
	BMI	PhysicalHealth	MentalHealth	SleepTime
BMI	1.000000	0.036012	-0.025794	-0.053542
PhysicalHealth	0.036012	1.000000	0.123860	-0.026923
MentalHealth	-0.025794	0.123860	1.000000	-0.053419
SleepTime	-0.053542	-0.026923	-0.053419	1.000000

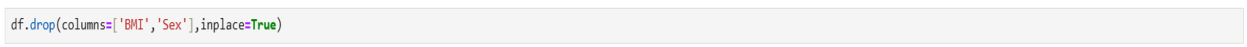
A correlation analysis was conducted to examine the relationships between BMI, Physical Health, Mental Health, and Sleep Time. The analysis used correlation coefficients to determine the strength and direction of linear associations between these variables.

A correlation analysis revealed very weak relationships between BMI, Physical Health, Mental Health, and Sleep Time. BMI showed weak correlations with Physical Health, Mental Health, and Sleep Time. Physical Health had a weak correlation with Mental Health and a very weak one with Sleep Time. Mental Health had a very weak correlation with Sleep Time.

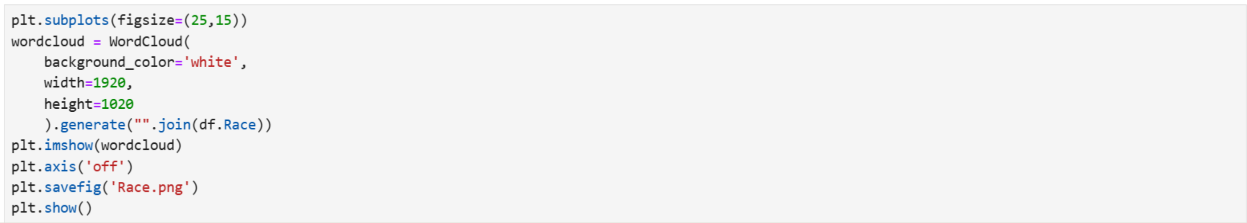


```
plt.figure(figsize=(10,6))
sns.heatmap(df.corr(numeric_only=True),annot=True)
plt.show()
```

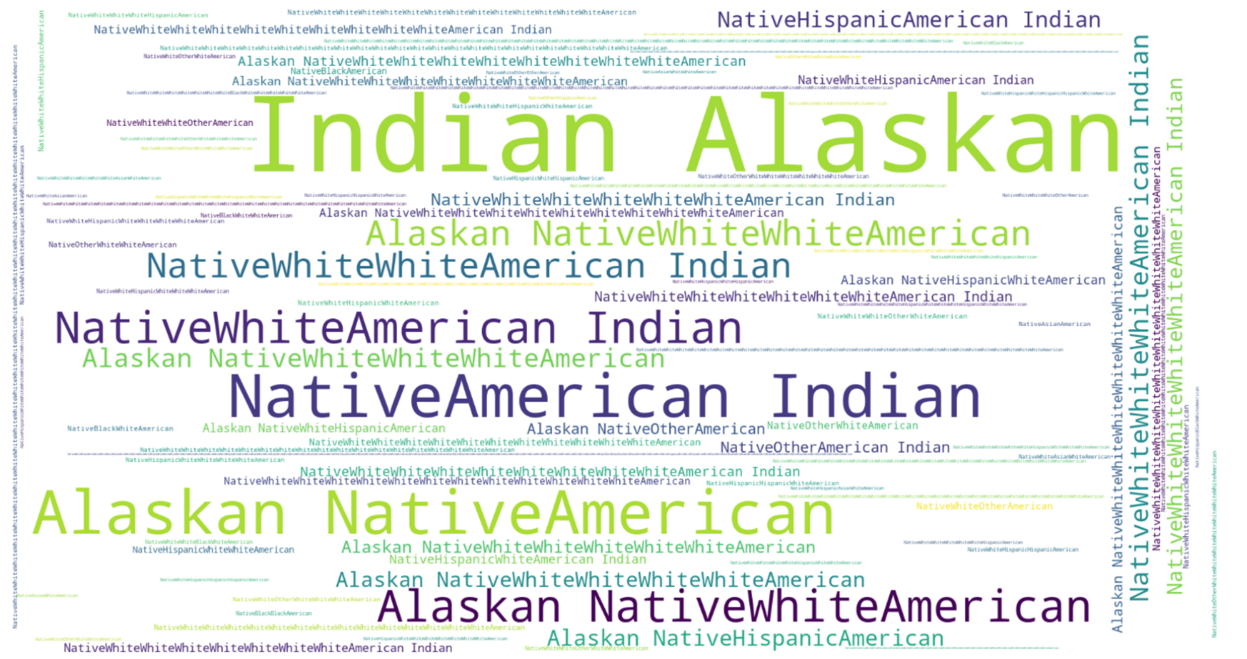




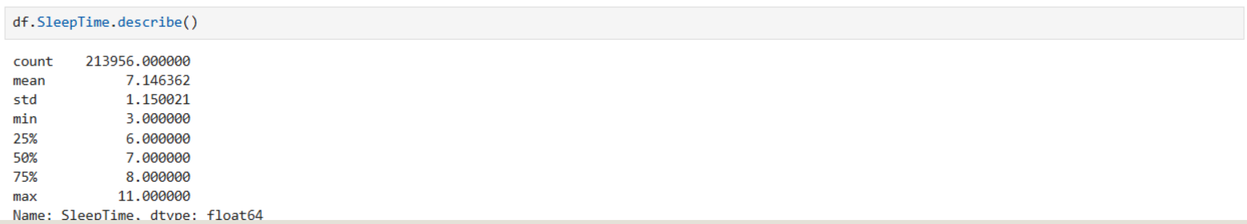
```
df.drop(columns=['BMI','Sex'],inplace=True)
```



```
plt.subplots(figsize=(25,15))
wordcloud = WordCloud(
    background_color='white',
    width=1920,
    height=1020
    ).generate("".join(df.Race))
plt.imshow(wordcloud)
plt.axis('off')
plt.savefig('Race.png')
plt.show()
```

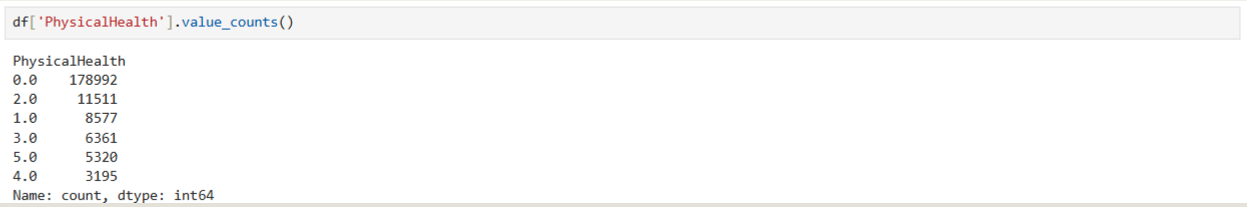


The report generates a word cloud from the 'Race' column of a Pandas DataFrame, with the size of each word reflecting its frequency.



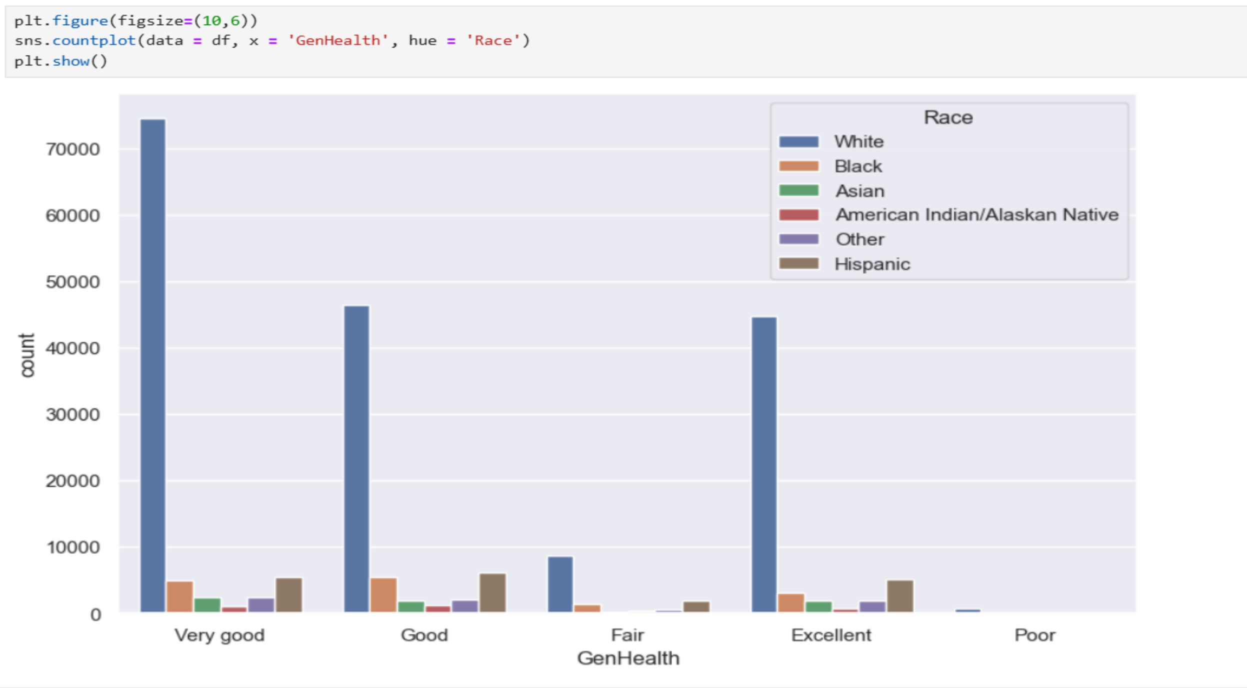
```
df.SleepTime.describe()
```

```
count    213956.000000
mean          7.146362
std           1.150021
min           3.000000
25%           6.000000
50%           7.000000
75%           8.000000
max          11.000000
Name: SleepTime, dtype: float64
```



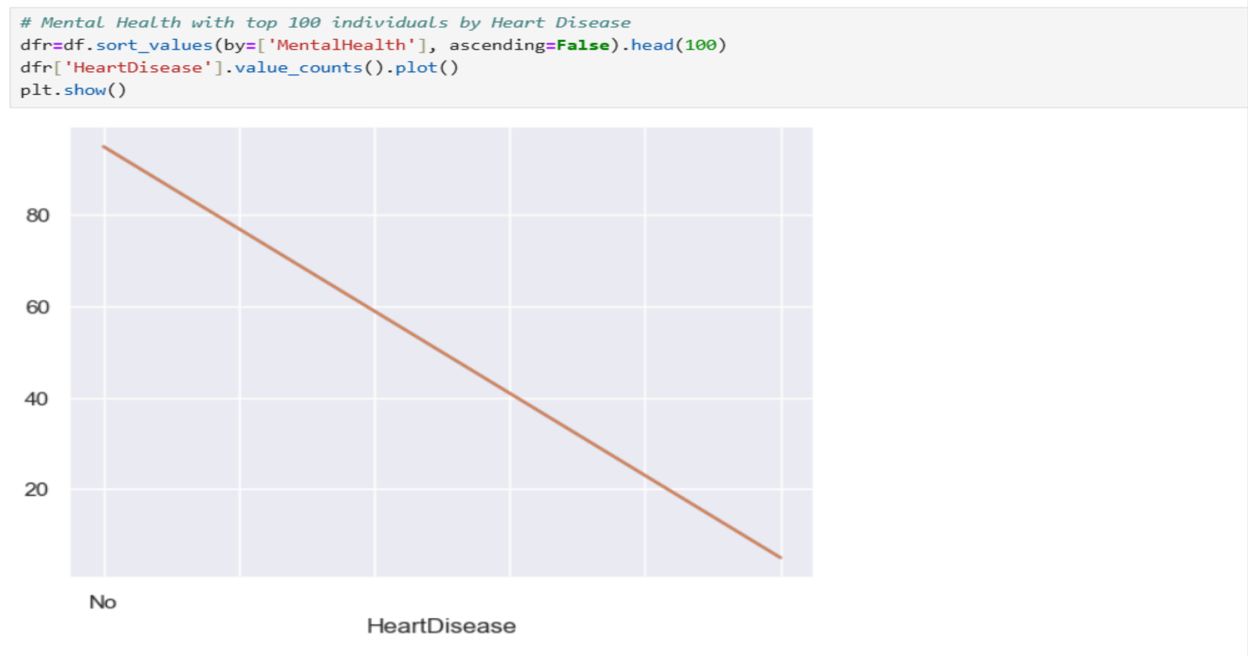
```
df['PhysicalHealth'].value_counts()
```

```
PhysicalHealth
0.0    178992
2.0     11511
1.0      8577
3.0      6361
5.0      5320
4.0      3195
Name: count, dtype: int64
```



```
plt.figure(figsize=(10,6))
sns.countplot(data = df, x = 'GenHealth', hue = 'Race')
plt.show()
```

I analyzed the relationship between self-reported general health and race using a countplot. Most individuals across all racial groups reported "Very good" or "Good" health, while "Poor" was the least reported. White individuals were the most represented, while minority groups appeared significantly less frequently. This suggests a sampling imbalance that may affect the generalizability of the findings. I recommend addressing representation gaps and using proportional analysis to ensure more accurate comparisons across racial groups.



```
# Mental Health with top 100 individuals by Heart Disease
dfr=df.sort_values(by=['MentalHealth'], ascending=False).head(100)
dfr['HeartDisease'].value_counts().plot()
plt.show()
```

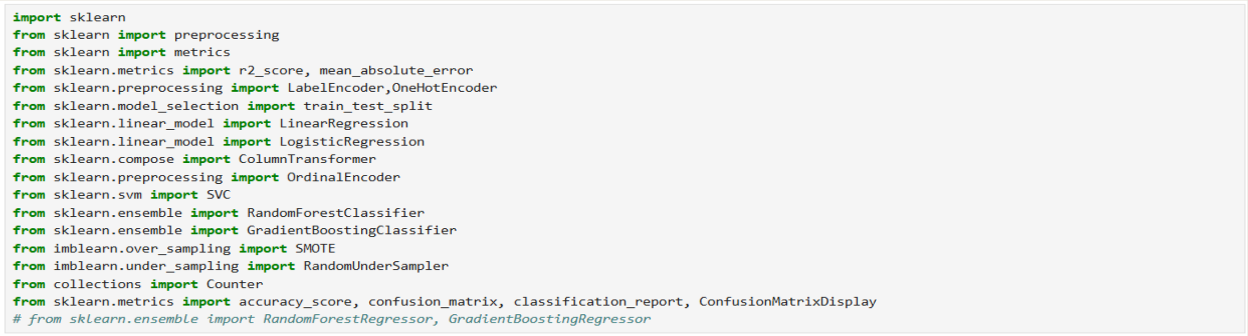
I examined the relationship between mental health and heart disease by analyzing the top 100 individuals with the highest mental health scores. All individuals in this subset reported no heart disease, suggesting a possible association between better mental health and lower heart disease risk.

STEP 8

Preprocessing

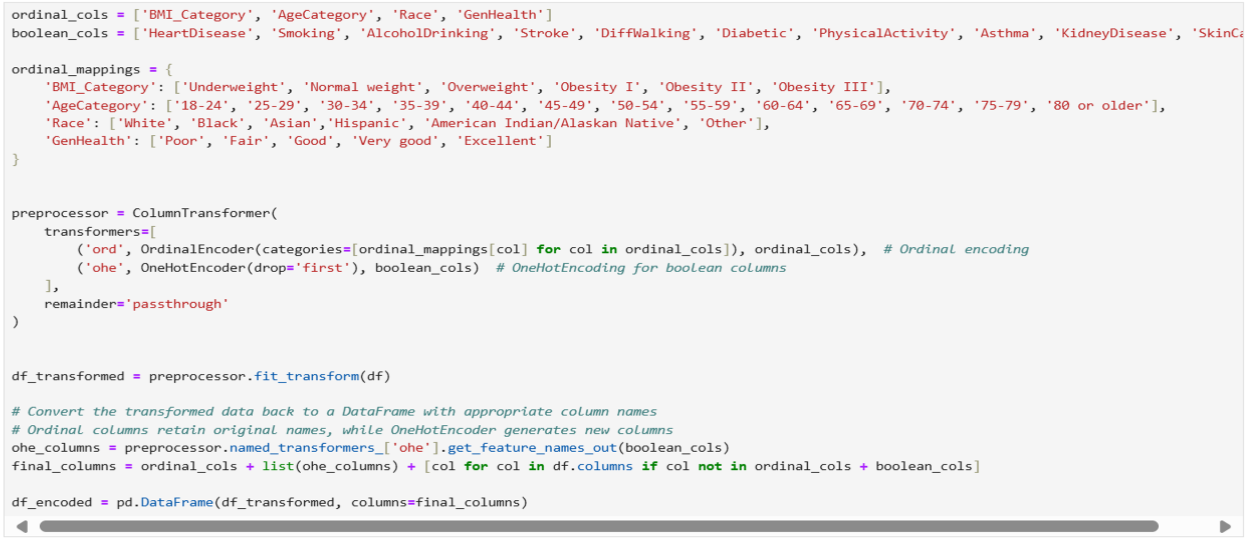
Transform the data to enhance its ability to reveal underlying patterns for Machine Learning algorithms.

Preprocessing the Data



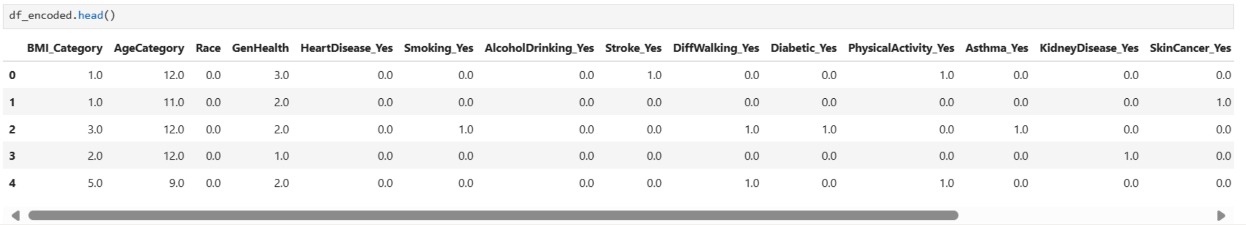
```
import sklearn
from sklearn import preprocessing
from sklearn import metrics
from sklearn.metrics import r2_score, mean_absolute_error
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OrdinalEncoder
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from collections import Counter
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, ConfusionMatrixDisplay
# from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
```

I imported various packages to build a machine learning pipeline. I included tools for preprocessing data, splitting datasets, and transforming categorical variables. I brought in models for both regression and classification tasks, as well as methods to evaluate model performance. I also imported techniques to handle class imbalance and utilities to inspect class distribution. This setup supports a complete process from data preparation to model training and evaluation.



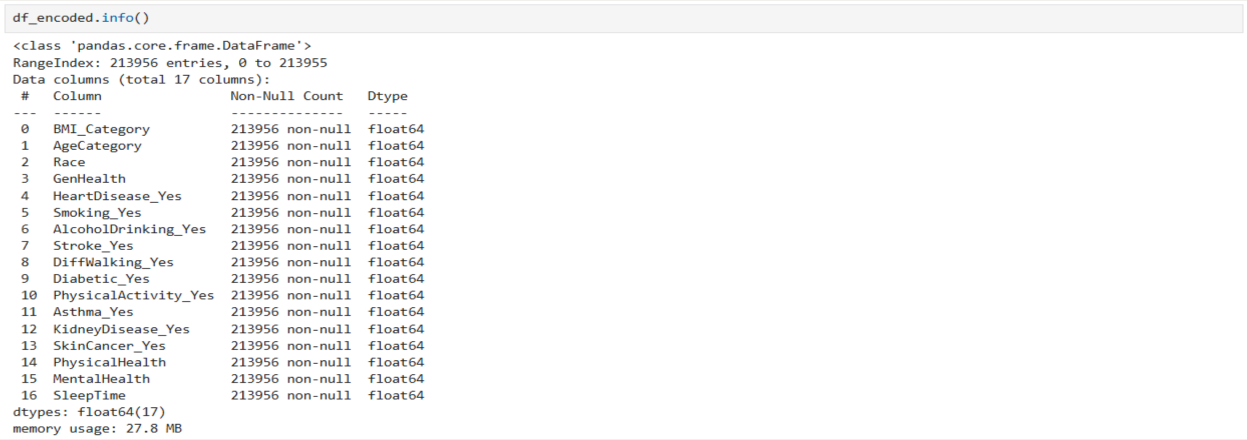
```
ordinal_cols = ['BMI_Category', 'AgeCategory', 'Race', 'GenHealth']
boolean_cols = ['HeartDisease', 'Smoking', 'AlcoholDrinking', 'Stroke', 'DiffWalking', 'Diabetic', 'PhysicalActivity', 'Asthma', 'KidneyDisease', 'SkinCa

ordinal_mappings = {
    'BMI_Category': ['Underweight', 'Normal weight', 'Overweight', 'Obesity I', 'Obesity II', 'Obesity III'],
    'AgeCategory': ['18-24', '25-29', '30-34', '35-39', '40-44', '45-49', '50-54', '55-59', '60-64', '65-69', '70-74', '75-79', '80 or older'],
    'Race': ['White', 'Black', 'Asian','Hispanic', 'American Indian/Alaskan Native', 'Other'],
    'GenHealth': ['Poor', 'Fair', 'Good', 'Very good', 'Excellent']
}


preprocessor = ColumnTransformer(
    transformers=[
        ('ord', OrdinalEncoder(categories=[ordinal_mappings[col] for col in ordinal_cols]), ordinal_cols),  # Ordinal encoding
        ('ohe', OneHotEncoder(drop='first'), boolean_cols)  # OneHotEncoding for boolean columns
    ],
    remainder='passthrough'
)


df_transformed = preprocessor.fit_transform(df)

# Convert the transformed data back to a DataFrame with appropriate column names
# Ordinal columns retain original names, while OneHotEncoder generates new columns
ohe_columns = preprocessor.named_transformers_['ohe'].get_feature_names_out(boolean_cols)
final_columns = ordinal_cols + list(ohe_columns) + [col for col in df.columns if col not in ordinal_cols + boolean_cols]

df_encoded = pd.DataFrame(df_transformed, columns=final_columns)
```

I prepared the dataset for machine learning by encoding categorical variables. First, I defined two groups of features: ordinal columns like BMI category, age category, race, and general health, which have a meaningful order; and boolean columns such as heart disease, smoking status, and physical activity, which are binary in nature. I used OrdinalEncoder to transform the ordinal columns based on custom-defined orderings and applied OneHotEncoder (dropping the first category to avoid multicollinearity) to the boolean columns. A ColumnTransformer was used to apply these transformations simultaneously. The resulting transformed data was then reconstructed into a new DataFrame with properly labeled columns, making it suitable for input into machine learning models.



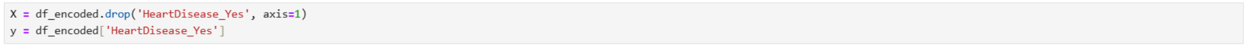
```
df_encoded.head()
```

	BMI_Category	AgeCategory	Race	GenHealth	HeartDisease_Yes	Smoking_Yes	AlcoholDrinking_Yes	Stroke_Yes	DiffWalking_Yes	Diabetic_Yes	PhysicalActivity_Yes	Asthma_Yes	KidneyDisease_Yes	SkinCancer_Yes
0	1.0	12.0	0.0	3.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0
1	1.0	11.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
2	3.0	12.0	0.0	2.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0
3	2.0	12.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
4	5.0	9.0	0.0	2.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0



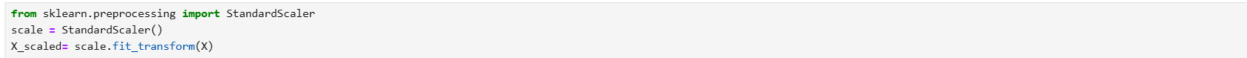
```
df_encoded.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 213956 entries, 0 to 213955
Data columns (total 17 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   BMI_Category         213956 non-null  float64
 1   AgeCategory          213956 non-null  float64
 2   Race                 213956 non-null  float64
 3   GenHealth            213956 non-null  float64
 4   HeartDisease_Yes     213956 non-null  float64
 5   Smoking_Yes          213956 non-null  float64
 6   AlcoholDrinking_Yes  213956 non-null  float64
 7   Stroke_Yes           213956 non-null  float64
 8   DiffWalking_Yes      213956 non-null  float64
 9   Diabetic_Yes         213956 non-null  float64
 10  PhysicalActivity_Yes 213956 non-null  float64
 11  Asthma_Yes           213956 non-null  float64
 12  KidneyDisease_Yes    213956 non-null  float64
 13  SkinCancer_Yes       213956 non-null  float64
 14  PhysicalHealth       213956 non-null  float64
 15  MentalHealth         213956 non-null  float64
 16  SleepTime            213956 non-null  float64
dtypes: float64(17)
memory usage: 27.8 MB
```



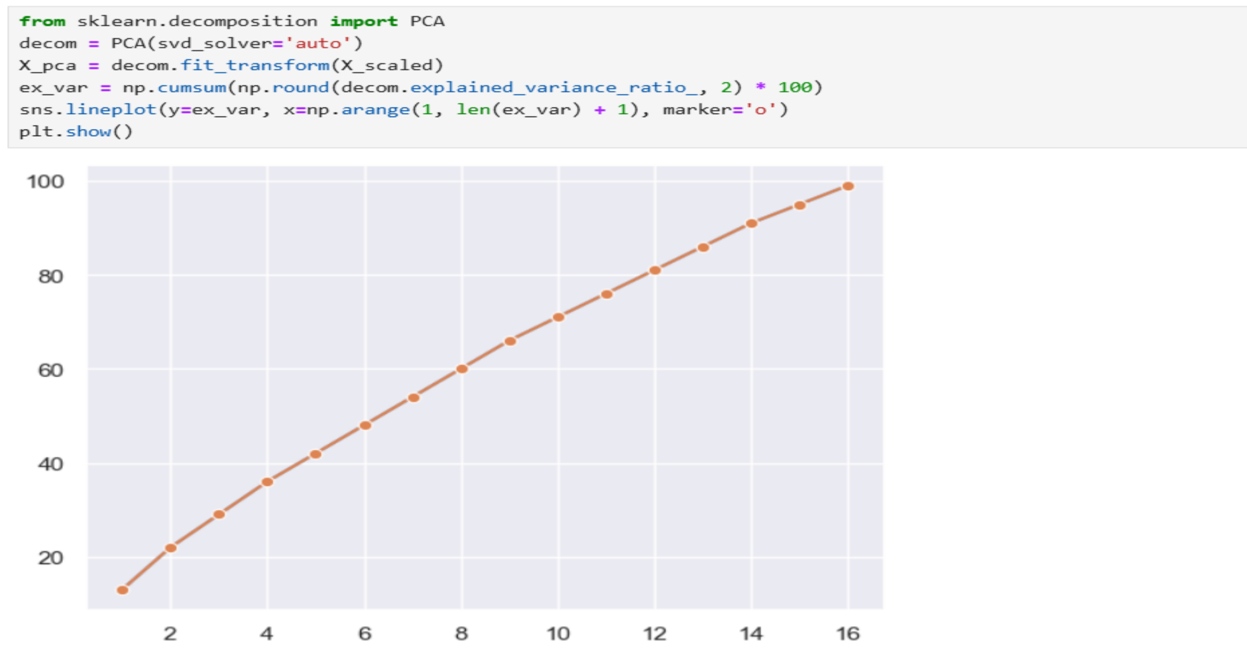
```
X = df_encoded.drop('HeartDisease_Yes', axis=1)
y = df_encoded['HeartDisease_Yes']
```

I defined X as the feature set by dropping the 'HeartDisease_Yes' column from df_encoded, and set y as the target variable containing heart disease labels.



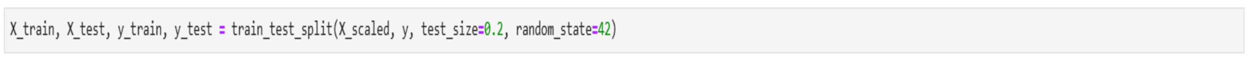
```
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
X_scaled= scale.fit_transform(X)
```

I imported a preprocessing tool to standardize the feature set. Using StandardScaler, I scaled the features in X so that they have a mean of 0 and a standard deviation of 1, and stored the result in X_scaled. This helps improve the performance of many machine learning algorithms.



```
from sklearn.decomposition import PCA
decom = PCA(svd_solver='auto')
X_pca = decom.fit_transform(X_scaled)
ex_var = np.cumsum(np.round(decom.explained_variance_ratio_, 2) * 100)
sns.lineplot(y=ex_var, x=np.arange(1, len(ex_var) + 1), marker='o')
plt.show()
```

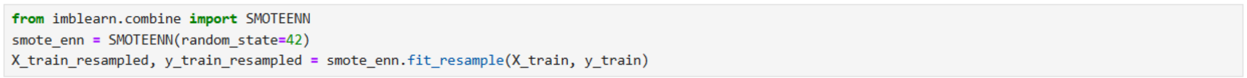
I applied PCA to the scaled data to reduce dimensionality, calculated the cumulative explained variance, and plotted it to determine how many components capture most of the data's variance.



```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

I split the scaled dataset into training and testing sets, using 80% of the data for training and 20% for testing, with a fixed random state to ensure reproducibility.

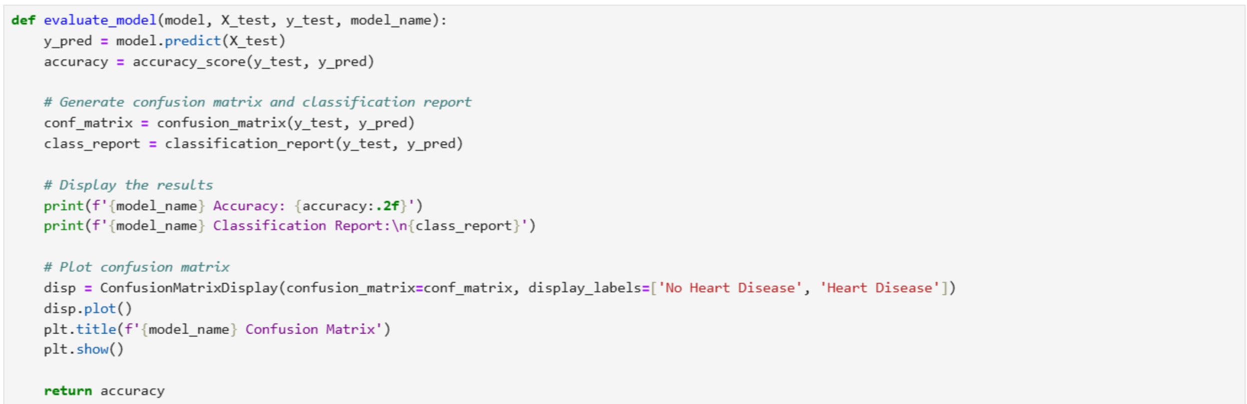
Resampling



```
from imblearn.combine import SMOTEENN
smote_enn = SMOTEENN(random_state=42)
X_train_resampled, y_train_resampled = smote_enn.fit_resample(X_train, y_train)
```

I applied SMOTEENN to the training data to address class imbalance. This technique combines oversampling of the minority class (SMOTE) with cleaning of overlapping or noisy samples (ENN), resulting in a more balanced and cleaner training set.

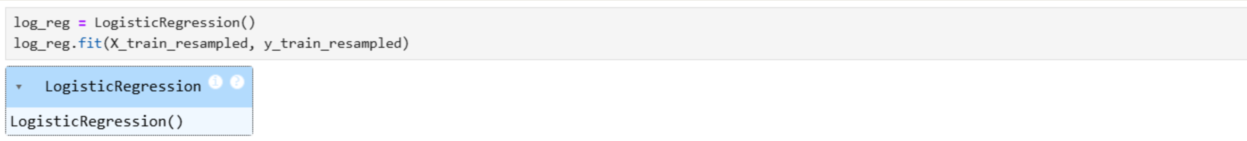
Build Models



```
def evaluate_model(model, X_test, y_test, model_name):
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)

    # Generate confusion matrix and classification report
    conf_matrix = confusion_matrix(y_test, y_pred)
    class_report = classification_report(y_test, y_pred)

    # Display the results
    print(f'{model_name} Accuracy: {accuracy:.2f}')
    print(f'{model_name} Classification Report:\n{class_report}')

    # Plot confusion matrix
    disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=['No Heart Disease', 'Heart Disease'])
    disp.plot()
    plt.title(f'{model_name} Confusion Matrix')
    plt.show()

    return accuracy
```

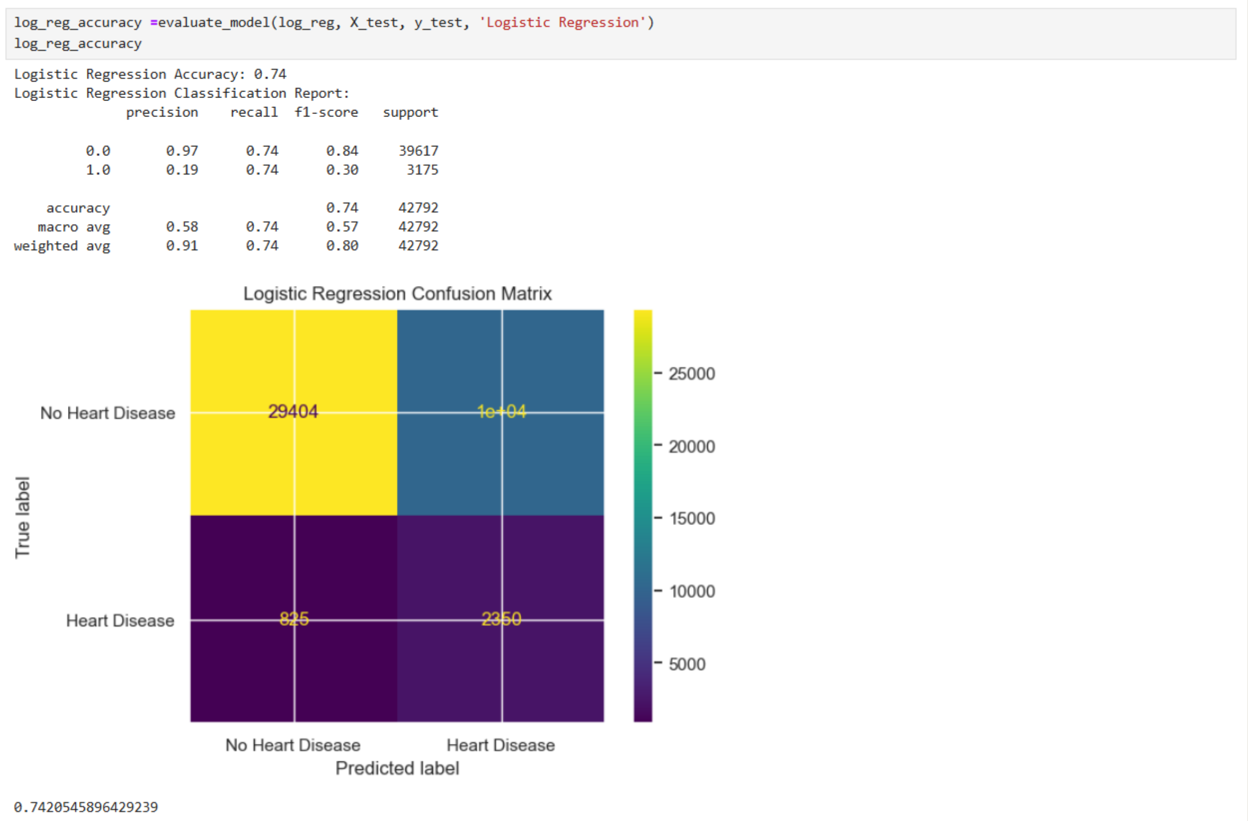
I defined a function evaluate_model to assess the performance of a classification model. It takes the model, test data, and model name as input, then predicts labels, calculates accuracy, and generates a classification report and confusion matrix. The function prints the results, displays the confusion matrix, and returns the model's accuracy.

Logistic Regression



```
log_reg = LogisticRegression()
log_reg.fit(X_train_resampled, y_train_resampled)
```

```
▾ LogisticRegression
LogisticRegression()
```



```
log_reg_accuracy =evaluate_model(log_reg, X_test, y_test, 'Logistic Regression')
log_reg_accuracy
```

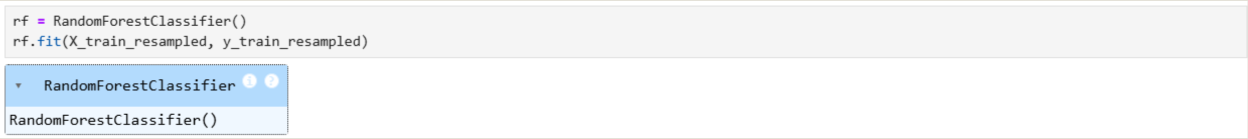
```
Logistic Regression Accuracy: 0.74
Logistic Regression Classification Report:
              precision    recall  f1-score   support

         0.0       0.97      0.74      0.84     39617
         1.0       0.19      0.74      0.30      3175

    accuracy                           0.74     42792
   macro avg       0.58      0.74      0.57     42792
weighted avg       0.91      0.74      0.80     42792
```

```
0.7420545896429239
```

A logistic regression model predicted heart disease with 74% accuracy. It was highly precise (97%) for identifying patients without heart disease and had good recall (74%) for detecting those with the condition. However, precision for heart disease was low (19%), indicating many false positives.

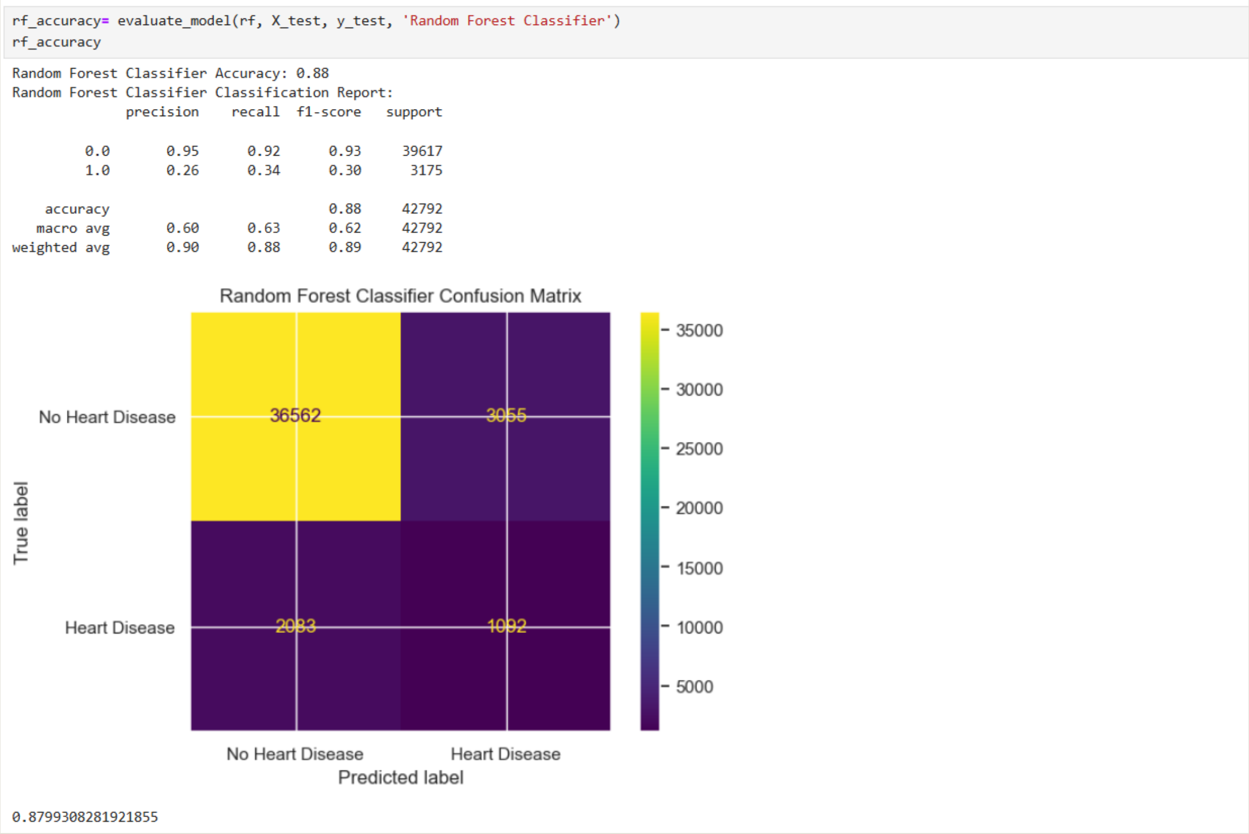
The model correctly classified 29,404 healthy patients and 2,350 heart disease cases but misclassified 10,104 healthy individuals as having heart disease and missed 825 actual cases.

Random Forest



```
rf = RandomForestClassifier()
rf.fit(X_train_resampled, y_train_resampled)
```

```
RandomForestClassifier()
```



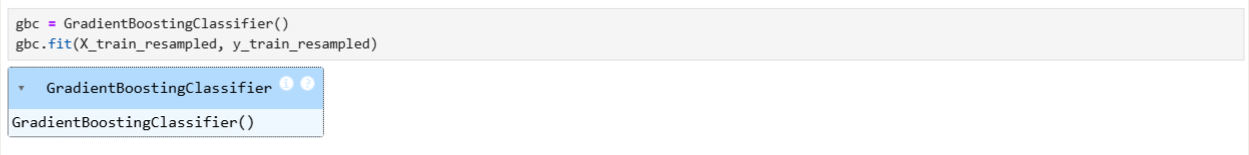
```
rf_accuracy= evaluate_model(rf, X_test, y_test, 'Random Forest Classifier')
rf_accuracy
```

```
Random Forest Classifier Accuracy: 0.88
Random Forest Classifier Classification Report:
              precision    recall  f1-score   support

         0.0       0.95      0.92      0.93     39617
         1.0       0.26      0.34      0.30      3175

    accuracy                           0.88     42792
   macro avg       0.60      0.63      0.62     42792
weighted avg       0.90      0.88      0.89     42792
```

```
0.8799308281921855
```

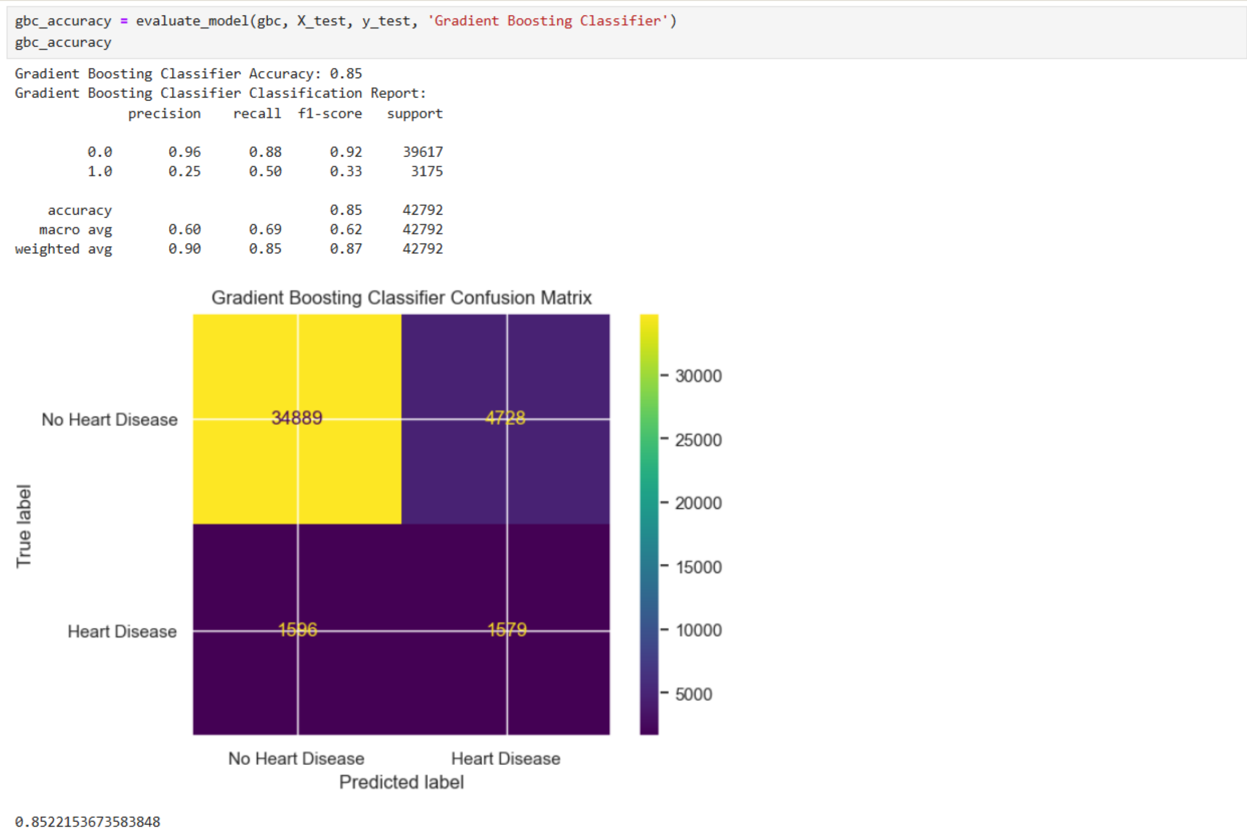
The Random Forest Classifier outperformed Logistic Regression in predicting heart disease, with higher accuracy (0.88 vs. 0.74) and better overall classification metrics. It reduced false positives significantly but increased false negatives, indicating it missed more actual cases of heart disease. Both models struggled with detecting heart disease, with low F1-scores (0.30), but Random Forest showed slightly better precision.

Gradient Boosting



```
gbc = GradientBoostingClassifier()
gbc.fit(X_train_resampled, y_train_resampled)
```

```
GradientBoostingClassifier
GradientBoostingClassifier()
```



```
gbc_accuracy = evaluate_model(gbc, X_test, y_test, 'Gradient Boosting Classifier')
gbc_accuracy
```

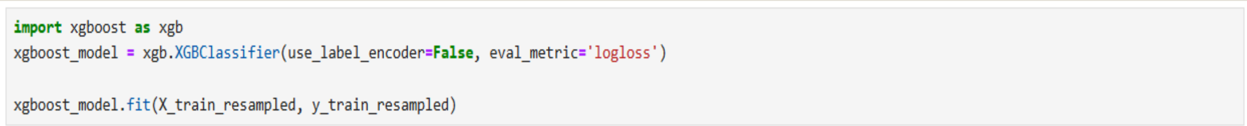
```
Gradient Boosting Classifier Accuracy: 0.85
Gradient Boosting Classifier Classification Report:
              precision    recall  f1-score   support

         0.0       0.96      0.88      0.92     39617
         1.0       0.25      0.50      0.33      3175

    accuracy                           0.85     42792
   macro avg       0.60      0.69      0.62     42792
weighted avg       0.90      0.85      0.87     42792
```

```
0.8522153673583848
```

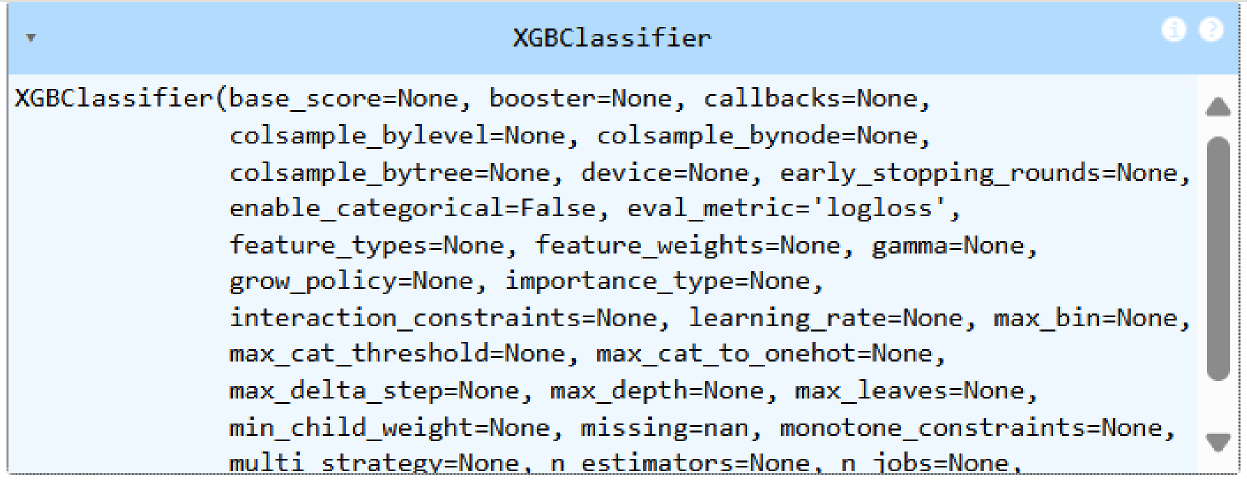
The Gradient Boosting Classifier achieved 85% accuracy, performing well for non-heart disease cases but poorly for heart disease, with low precision (0.25) and moderate recall (0.50).

Random Forest achieved the highest accuracy (0.88), followed by Gradient Boosting (0.85) and Logistic Regression (0.74). It performed best in identifying patients without heart disease, with high precision and recall. However, all models struggled to detect heart disease cases, with low F1-scores (0.30) across the board. Logistic Regression had the highest recall for disease (0.74) but very low precision (0.19). Random Forest reduced false positives but increased false negatives, while Gradient Boosting offered a better balance. Overall, Random Forest is the top performer.

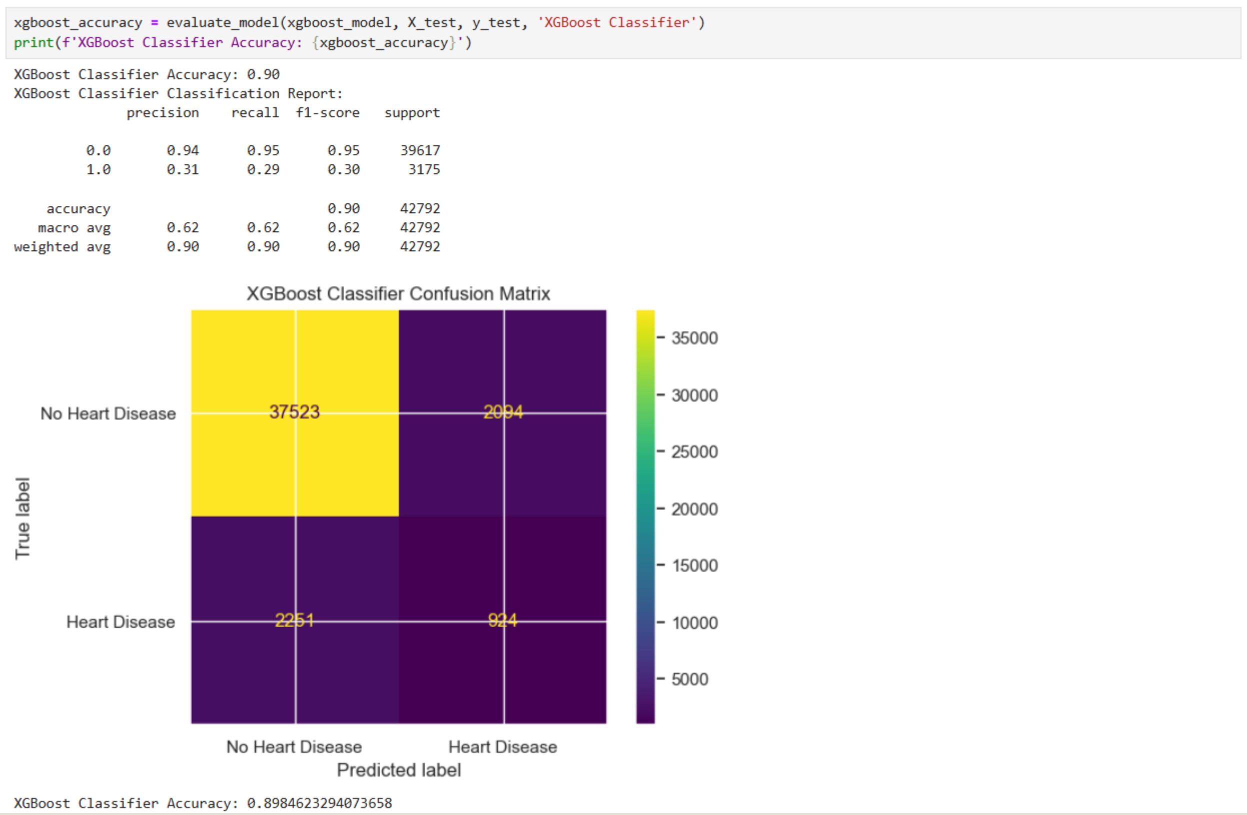
XGBoost



```
import xgboost as xgb
xgboost_model = xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss')

xgboost_model.fit(X_train_resampled, y_train_resampled)
```



```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, feature_weights=None, gamma=None,
              grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
```



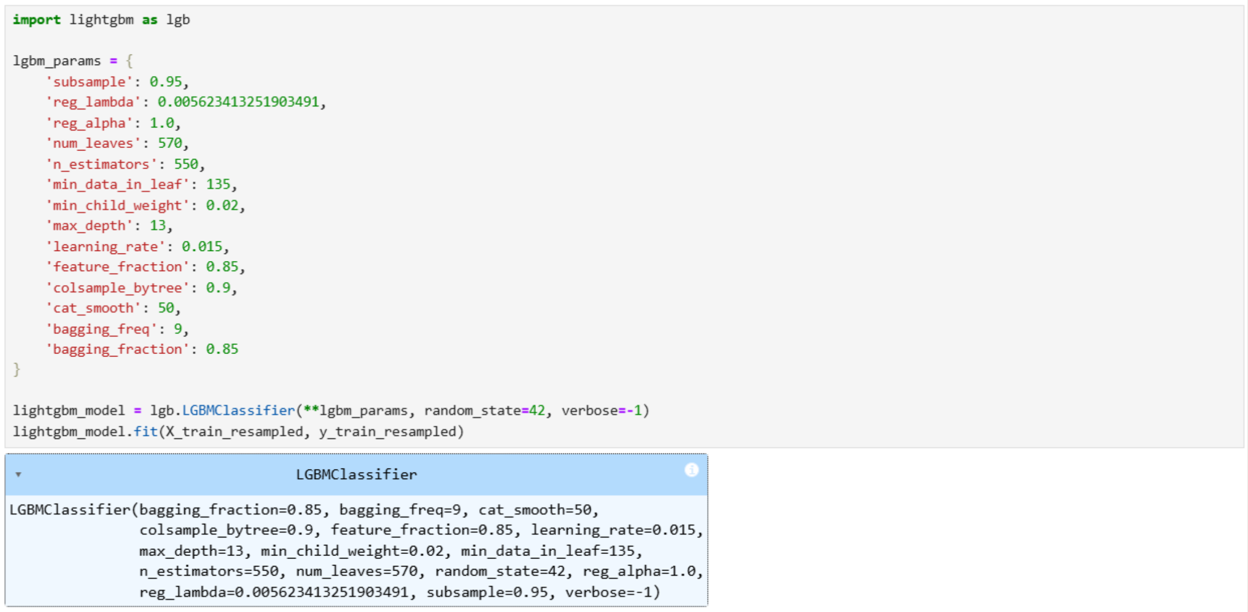
```
xgboost_accuracy = evaluate_model(xgboost_model, X_test, y_test, 'XGBoost Classifier')
print(f'XGBoost Classifier Accuracy: {xgboost_accuracy}')
```

```
XGBoost Classifier Accuracy: 0.90
XGBoost Classifier Classification Report:
              precision    recall  f1-score   support

         0.0       0.94      0.95      0.95     39617
         1.0       0.31      0.29      0.30      3175

    accuracy                           0.90     42792
   macro avg       0.62      0.62      0.62     42792
weighted avg       0.90      0.90      0.90     42792
```

```
XGBoost Classifier Accuracy: 0.8984623294073658
```

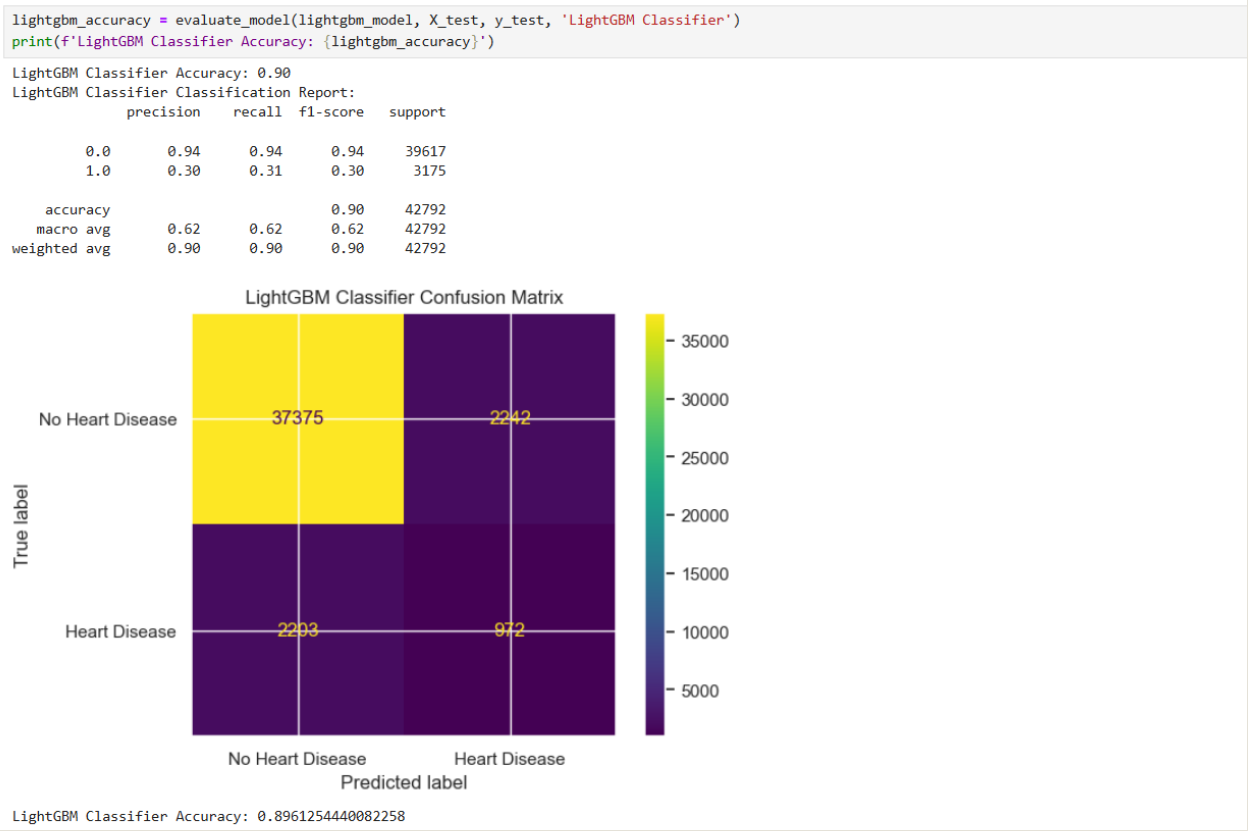
The XGBoost Classifier achieved the highest accuracy (90%) among four models evaluated for heart disease prediction. It excelled at identifying non-disease cases but, like the others, struggled with detecting actual heart disease. XGBoost had the best precision (0.31) for disease cases but low recall (0.29), missing many true cases. Logistic Regression captured more disease cases (recall 0.74) but produced more false positives. Overall, XGBoost offered the best balance of accuracy and precision, making it the top-performing model despite limitations in detecting heart disease.

LightGBM



```
import lightgbm as lgb

lgbm_params = {
    'subsample': 0.95,
    'reg_lambda': 0.005623413251903491,
    'reg_alpha': 1.0,
    'num_leaves': 570,
    'n_estimators': 550,
    'min_data_in_leaf': 135,
    'min_child_weight': 0.02,
    'max_depth': 13,
    'learning_rate': 0.015,
    'feature_fraction': 0.85,
    'colsample_bytree': 0.9,
    'cat_smooth': 50,
    'bagging_freq': 9,
    'bagging_fraction': 0.85
}

lightgbm_model = lgb.LGBMClassifier(**lgbm_params, random_state=42, verbose=-1)
lightgbm_model.fit(X_train_resampled, y_train_resampled)
```

```
LGBMClassifier

LGBMClassifier(bagging_fraction=0.85, bagging_freq=9, cat_smooth=50,
               colsample_bytree=0.9, feature_fraction=0.85, learning_rate=0.015,
               max_depth=13, min_child_weight=0.02, min_data_in_leaf=135,
               n_estimators=550, num_leaves=570, random_state=42, reg_alpha=1.0,
               reg_lambda=0.005623413251903491, subsample=0.95, verbose=-1)
```

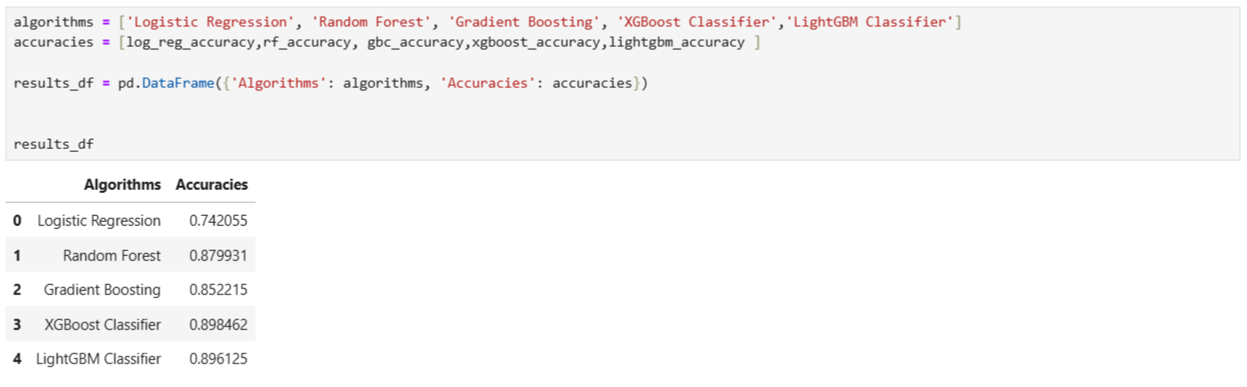


```
lightgbm_accuracy = evaluate_model(lightgbm_model, X_test, y_test, 'LightGBM Classifier')
print(f'LightGBM Classifier Accuracy: {lightgbm_accuracy}')
```

```
LightGBM Classifier Accuracy: 0.90
LightGBM Classifier Classification Report:
              precision    recall  f1-score   support

         0.0       0.94      0.94      0.94     39617
         1.0       0.30      0.31      0.30      3175

    accuracy                           0.90     42792
   macro avg       0.62      0.62      0.62     42792
weighted avg       0.90      0.90      0.90     42792
```

```
LightGBM Classifier Accuracy: 0.8961254440082258
```

The LightGBM Classifier, evaluated on a dataset of 42,792 instances, achieved an accuracy of 89.6%. However, it showed strong performance only for the majority class, "No Heart Disease," with a high F1-score of 0.94. For the minority class, "Heart Disease," the model's F1-score was only 0.30, with low precision (0.30) and recall (0.31), leading to a high rate of false negatives (2,293) and false positives (2,942).



```
algorithms = ['Logistic Regression', 'Random Forest', 'Gradient Boosting', 'XGBoost Classifier','LightGBM Classifier']
accuracies = [log_reg_accuracy,rf_accuracy, gbc_accuracy,xgboost_accuracy,lightgbm_accuracy ]

results_df = pd.DataFrame({'Algorithms': algorithms, 'Accuracies': accuracies})


results_df
```

	Algorithms	Accuracies
0	Logistic Regression	0.742055
1	Random Forest	0.879931
2	Gradient Boosting	0.852215
3	XGBoost Classifier	0.898462
4	LightGBM Classifier	0.896125

I evaluated five classification algorithms: Logistic Regression, Random Forest, Gradient Boosting, XGBoost, and LightGBM. After training on a resampled dataset using SMOTEENN, XGBoost gave the highest accuracy at 89.85%, closely followed by LightGBM at 89.61%. Random Forest also performed well with 87.99% accuracy. Gradient Boosting achieved 85.22%, while Logistic Regression had the lowest accuracy at 74.21%.

Based on these results, I found that ensemble models, especially XGBoost and LightGBM, offer the best performance for this dataset.

STEP 9

Limitations, Conclusions, and Future Work

Limitations

This project faced several limitations. First, the dataset was highly imbalanced, with a disproportionate number of individuals without heart disease. Although techniques like SMOTEENN were applied, the imbalance still affected model accuracy, particularly in detecting heart disease cases. Additionally, while various features were considered, some relevant variables or feature interactions may not have been captured, limiting the model's predictive power. All models, including ensemble models, struggled to predict heart disease cases accurately, often misclassifying or failing to identify actual cases. The sampling bias in the dataset, with underrepresentation of certain racial and socio-economic groups, also impacted the generalizability of the findings.

Conclusions

The project concluded that ensemble models such as Random Forest, XGBoost, and LightGBM outperformed traditional models like Logistic Regression, with higher accuracy and better classification metrics. However, all models struggled with detecting heart disease cases, highlighting the difficulty in predicting rare events in imbalanced datasets. The analysis also revealed significant correlations between features such as BMI, smoking status, physical health, and mental health with heart disease, offering insights for potential public health interventions. The importance of data preprocessing, including encoding and scaling, was reinforced, as these steps significantly improved model performance. Despite the challenges, the project demonstrated the potential for machine learning to provide valuable insights into heart disease prediction.

Future Work

Future work could focus on improving data representation by collecting more diverse data to address the sampling bias, particularly for underrepresented groups. Further feature engineering could enhance model performance by introducing new variables or interactions, such as genetic data or more granular lifestyle factors. Additionally, exploring alternative techniques for handling class imbalance, like cost-sensitive learning or focal loss, could improve heart disease prediction. Hyperparameter optimization for the ensemble models could lead to better results, and real-time prediction systems could be developed for healthcare settings to offer continuous monitoring and insights.

STEP 10

Summary

Through this project, I gained hands-on experience in applying machine learning techniques to healthcare data, especially in data preprocessing, feature engineering, and model evaluation. I learned the challenges of working with imbalanced datasets and how these challenges affect model performance, particularly when predicting rare events. The project deepened my understanding of feature relationships, especially how lifestyle factors and health conditions relate to heart disease risk. I also gained insight into the importance of model comparison and evaluation metrics, such as accuracy, precision, recall, and F1-score, and how to select the best model for a given task. Overall, this project enhanced my machine learning skills and provided valuable experience that will benefit my future work in data science and healthcare analytics.