In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import scipy.stats as stats
import statsmodels.api as sm

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, roc_


import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
df = pd.read_csv("/content/heart_2022_Key_indicators.csv")
```

In [3]:

```
df.head()
```

Out[3]:

	HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	Diff
0	No	16.60	Yes	No	No	3.0	30.0	
1	No	20.34	No	No	Yes	0.0	0.0	
2	No	26.58	Yes	No	No	20.0	30.0	
3	No	24.21	No	No	No	0.0	0.0	
4	No	23.71	No	No	No	28.0	0.0	

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 79649 entries, 0 to 79648
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   HeartDisease      79649 non-null  object 
 1   BMI               79649 non-null  float64
 2   Smoking           79649 non-null  object 
 3   AlcoholDrinking   79649 non-null  object 
 4   Stroke            79649 non-null  object 
 5   PhysicalHealth    79649 non-null  float64
 6   MentalHealth      79649 non-null  float64
 7   DiffWalking       79649 non-null  object 
 8   Sex               79649 non-null  object 
 9   AgeCategory       79649 non-null  object 
 10  Race              79648 non-null  object 
 11  Diabetic          79648 non-null  object 
 12  PhysicalActivity  79648 non-null  object 
 13  GenHealth         79648 non-null  object 
 14  SleepTime         79648 non-null  float64
 15  Asthma            79648 non-null  object 
 16  KidneyDisease     79648 non-null  object 
 17  SkinCancer        79648 non-null  object 
dtypes: float64(4), object(14)
memory usage: 10.9+ MB
```

In [5]:

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

Out[5]:

```
HeartDisease        0
BMI                 0
Smoking             0
AlcoholDrinking     0
Stroke              0
PhysicalHealth      0
MentalHealth        0
DiffWalking         0
Sex                 0
AgeCategory         0
Race                1
Diabetic            1
PhysicalActivity    1
GenHealth           1
SleepTime           1
Asthma              1
KidneyDisease       1
SkinCancer          1
dtype: int64
```

In [6]:

```
df = df.dropna(axis=0, how='any')
```

In [7]:

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

Out[7]:

```
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
```

In [8]:

```
df.describe()
```

Out[8]:

	BMI	PhysicalHealth	MentalHealth	SleepTime
count	79648.000000	79648.000000	79648.000000	79648.000000
mean	28.039626	3.503277	3.911109	7.101823
std	6.305998	8.077699	7.961594	1.497506
min	12.020000	0.000000	0.000000	1.000000
25%	23.710000	0.000000	0.000000	6.000000
50%	27.070000	0.000000	0.000000	7.000000
75%	31.180000	2.000000	3.000000	8.000000
max	87.050000	30.000000	30.000000	24.000000

In [9]:

```
df.shape
```

Out[9]:

```
(79648, 18)
```

🔷Basic EDA¶

- There are 17 predictor variables and 1 target variable.
- Out of the 17 predictor variables, 4 are numeric.
- There are 319795 observations.
- No variables have missing values.
- The numeric variables are all right skewed.

This means that their distribution is not normal, and they have a longer tail on the right side. This can affect the accuracy of some statistical models, such as linear regression, which assume normality in the distribution of the predictor variables.

In [10]:

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

Out[10]:

```
65-69          8328
60-64          8084
70-74          7984
55-59          7145
80 or older    6349
50-54          6268
75-79          5770
45-49          5381
35-39          5090
40-44          5085
18-24          5051
30-34          4841
25-29          4272
Name: AgeCategory, dtype: int64
```

In [11]:

```
desc = pd.DataFrame(df.describe(include = 'all').transpose())

def summary_stats(df):
    print(f'The shape of the data is: {df.shape}')
    summary = pd.DataFrame(df.dtypes, columns = ['data type'])
    summary['Number of missing values'] = df.isnull().sum().values
    summary['% of missing values'] = df.isnull().sum().values / len(df) * 100
    summary['min value'] = desc['min'].values
    summary['mean value'] = desc['mean'].values
    summary['max value'] = desc['max'].values
    return summary
```

👁 Summary Table

Below table shows the number of missing values for each variable as well as the minimum and maximum value for each variable.

In [12]:

```
summary_stats(df)
```

```
The shape of the data is: (79648, 18)
```

Out[12]:

	data type	Number of missing values	% of missing values	min value	mean value	max value
HeartDisease	object	0	0.0	NaN	NaN	NaN
BMI	float64	0	0.0	12.02	28.039626	87.05
Smoking	object	0	0.0	NaN	NaN	NaN
AlcoholDrinking	object	0	0.0	NaN	NaN	NaN
Stroke	object	0	0.0	NaN	NaN	NaN
PhysicalHealth	float64	0	0.0	0.0	3.503277	30.0
MentalHealth	float64	0	0.0	0.0	3.911109	30.0
DiffWalking	object	0	0.0	NaN	NaN	NaN
Sex	object	0	0.0	NaN	NaN	NaN
AgeCategory	object	0	0.0	NaN	NaN	NaN
Race	object	0	0.0	NaN	NaN	NaN
Diabetic	object	0	0.0	NaN	NaN	NaN
PhysicalActivity	object	0	0.0	NaN	NaN	NaN
GenHealth	object	0	0.0	NaN	NaN	NaN
SleepTime	float64	0	0.0	1.0	7.101823	24.0
Asthma	object	0	0.0	NaN	NaN	NaN
KidneyDisease	object	0	0.0	NaN	NaN	NaN
SkinCancer	object	0	0.0	NaN	NaN	NaN

In [13]:

```python
skewness = df.skew()

skew_df = pd.DataFrame({'Variable': skewness.index, 'Skewness': skewness.values})
skew_df
```

Out[13]:

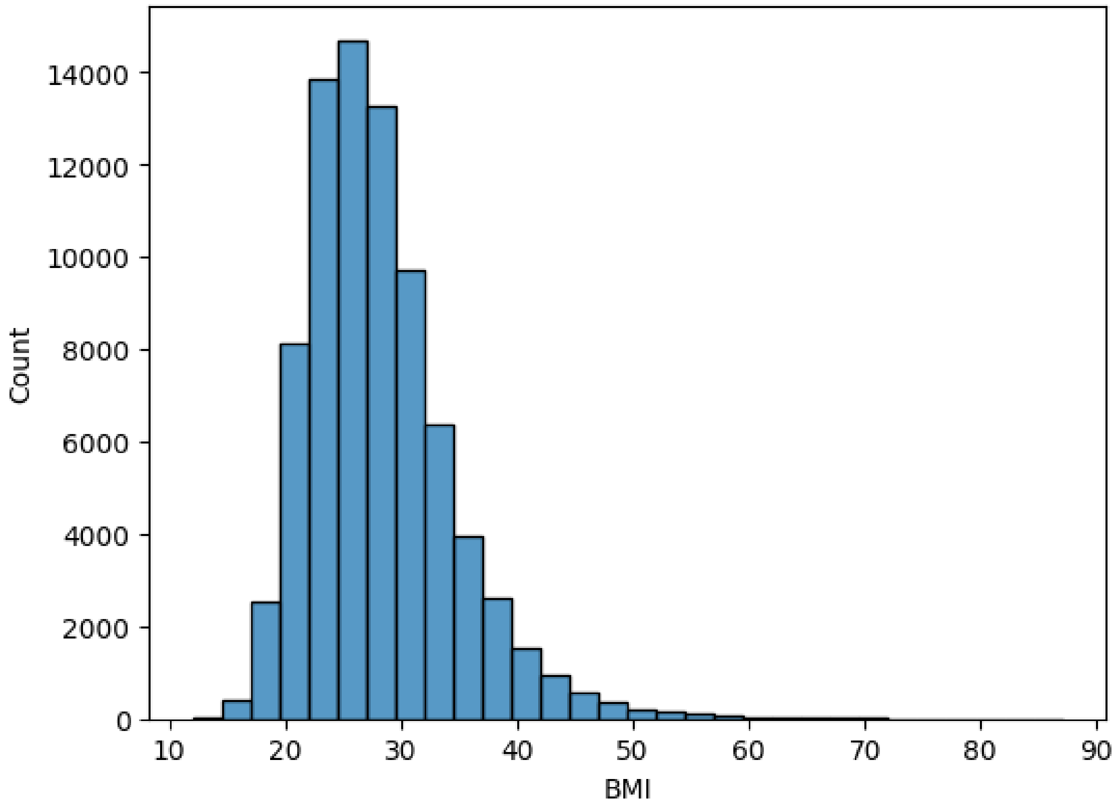
	Variable	Skewness
0	BMI	1.343807
1	PhysicalHealth	2.531653
2	MentalHealth	2.323737
3	SleepTime	0.994238

Since the values are all positive, the numeric variables are all right skewed.

👀Visualizations

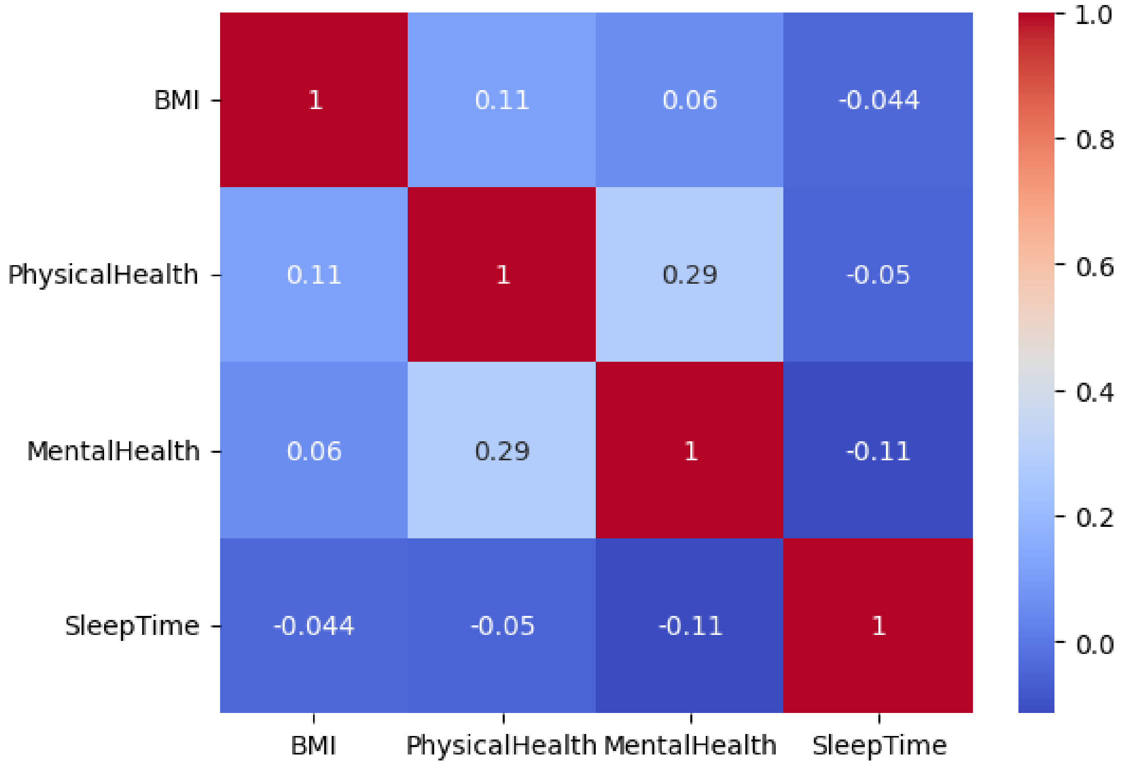
In [14]:

```python
sns.histplot(data=df, x='BMI', bins=30)
plt.show()
```



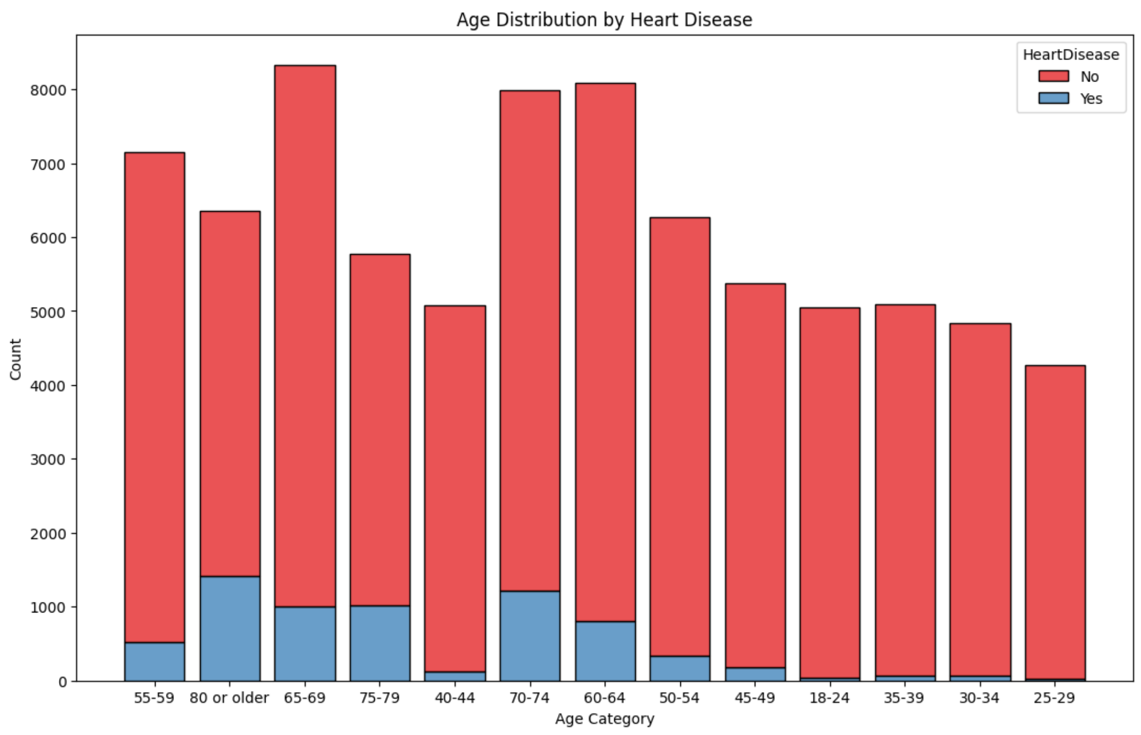
In [15]:

```python
corr_matrix = df.corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.show()
```



In [16]:

```
plt.figure(figsize=(13, 8))
sns.histplot(data=df, x='AgeCategory', hue='HeartDisease', multiple='stack', shrink=
plt.title('Age Distribution by Heart Disease')
plt.xlabel('Age Category')
plt.ylabel('Count')
plt.show()
```

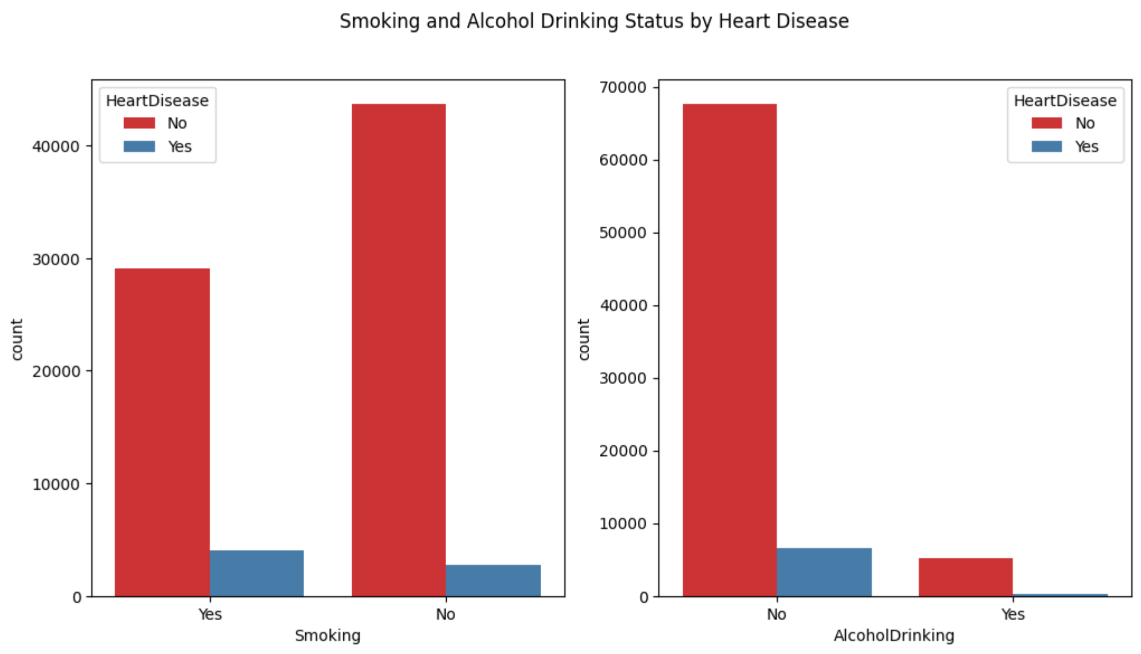


🔶 **The majority of the population is between 40 and 70 years old.**

🔶 **The highest frequency at around 60 years old.**

In [17]:

```python
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))
sns.countplot(data=df, x='Smoking', hue='HeartDisease', palette='Set1', ax=ax1)
sns.countplot(data=df, x='AlcoholDrinking', hue='HeartDisease', palette='Set1', ax=a
plt.suptitle('Smoking and Alcohol Drinking Status by Heart Disease')
plt.show()
```



🔶**In the first plot,**

☐**Individuals who smokes have a higher incidence of heart disease compared to those who do not smoke.**

☐**This highlights the potential negative impact of smoking on heart health.**

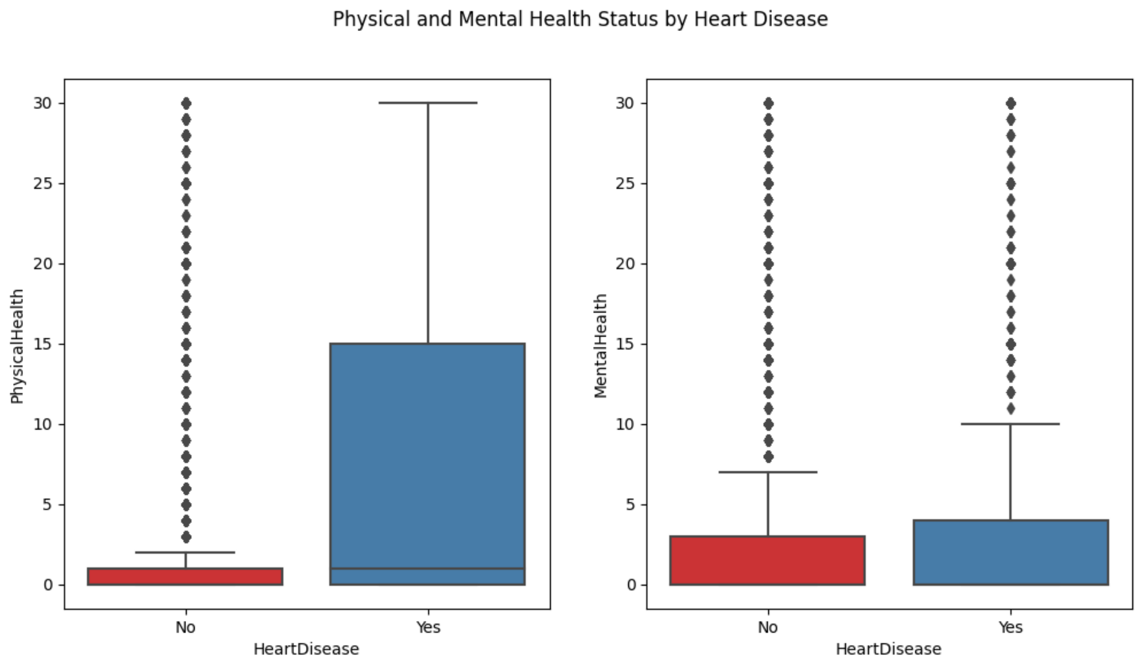
🔶**In the second plot,**

☐**Individuals who drink alcohol regularly have a slightly higher incidence of heart disease compared to those who do not drink alcohol.**

🔶**Overall, this graph suggests that smoking status may be a stronger predictor of heart disease compared to alcohol drinking status.**

In [18]:

```python
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))
sns.boxplot(data=df, x='HeartDisease', y='PhysicalHealth', palette='Set1', ax=ax1)
sns.boxplot(data=df, x='HeartDisease', y='MentalHealth', palette='Set1', ax=ax2)
plt.suptitle('Physical and Mental Health Status by Heart Disease')
plt.show()
```



🔶**The left boxplot shows**

□**Individuals with Heart Disease have a lower median Physical Health score compared to those without Heart Disease.**

🔶**The right boxplot shows**

□**Individuals with Heart Disease have a lower median Mental Health score compared to those without Heart Disease**

🔶**Overall, the boxplots suggest that individuals with Heart Disease tend to have lower Physical and Mental Health scores compared to those without Heart Disease.**

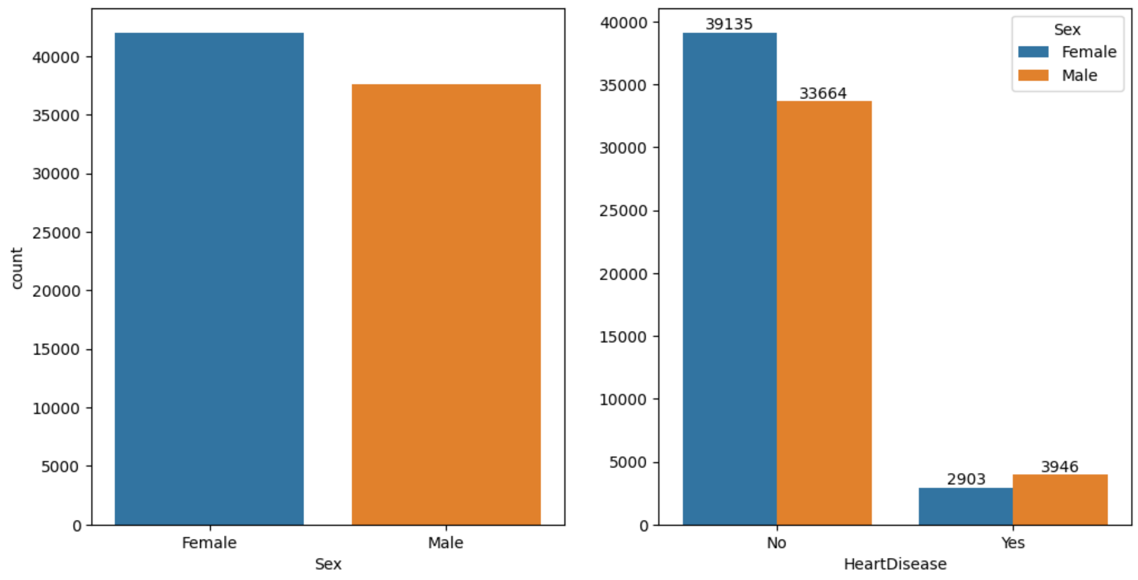
In [19]:

```python
fig, axs = plt.subplots(1, 2, figsize=(12, 6))

sns.countplot(x='Sex', data=df, ax=axs[0])
ax1 = sns.countplot(x='HeartDisease', hue='Sex', data=df, ax=axs[1])

for container in ax1.containers:
    ax1.bar_label(container)
    ax1.set_ylabel("")

plt.show()
```



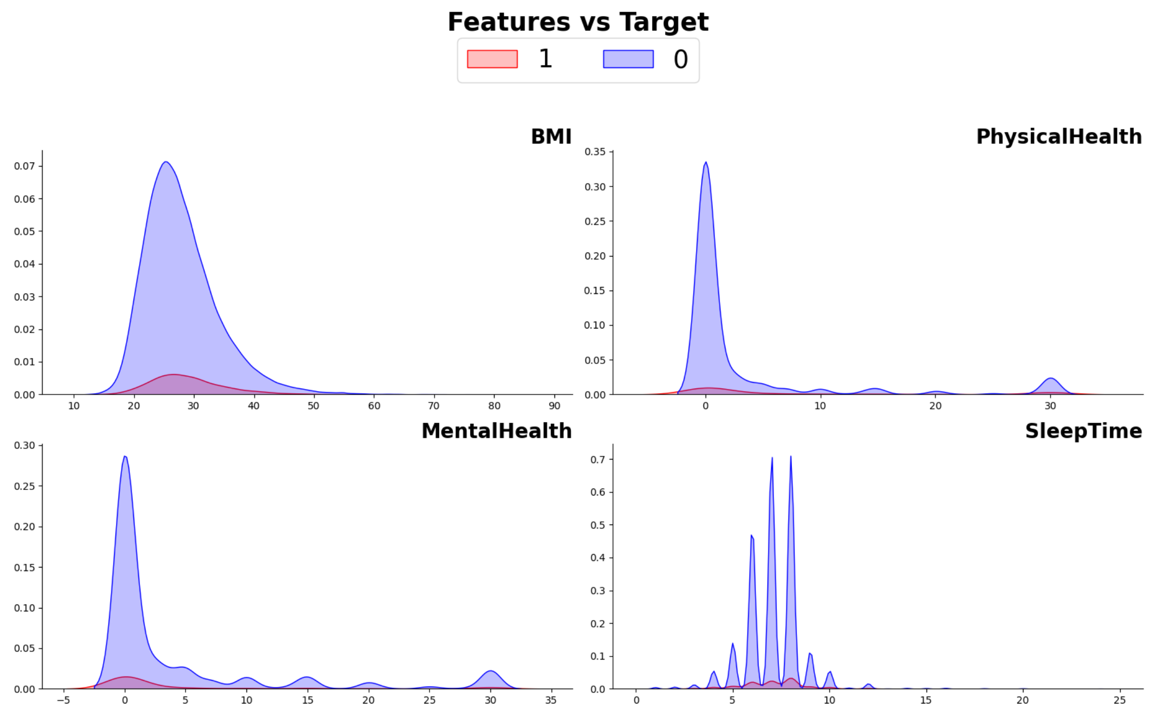
In [20]:

```
num_cols = df.select_dtypes(include=['float64','int64']).columns.tolist()

figsize = (16, 10)
fig = plt.figure(figsize=figsize)
for idx, col in enumerate(num_cols):
    ax = plt.subplot(2, 2, idx + 1)
    sns.kdeplot(
        data=df, hue='HeartDisease', fill=True,
        x=col, palette=['blue', 'red'], legend=False
    )

    ax.set_ylabel('')
    ax.spines['top'].set_visible(False)
    ax.set_xlabel('')
    ax.spines['right'].set_visible(False)
    ax.set_title(f'{col}', loc='right',
                 weight='bold', fontsize=20)

fig.suptitle(f'Features vs Target\n\n\n', ha='center',
             fontweight='bold', fontsize=25)
fig.legend([1, 0], loc='upper center', bbox_to_anchor=(0.5, 0.96), fontsize=25, ncol
plt.tight_layout()
plt.show()
```



🔷 Statistical Analysis

🔶 ANOVA

➖This test can be used to determine whether there is a significant difference between the means of three or more groups.

In [21]:

```
stats.shapiro(df['BMI'])
```

Out[21]:

```
ShapiroResult(statistic=0.9270806908607483, pvalue=0.0)
```

- Under a significance level of 0.05, BMI is not normally distributed and hence, ANOVA cannot be used.

🔶Chi-square test

➖This test can be used to determine whether there is a significant association between two categorical variables.

In [22]:

```
from scipy.stats import chi2_contingency

cont_table = pd.crosstab(df['HeartDisease'], df['Smoking'])

chi2_stat, p_val, dof, expected = chi2_contingency(cont_table)

print('Chi-square statistic:', chi2_stat)
print('p-value:', p_val)
```

```
Chi-square statistic: 992.8561974251235
p-value: 6.413227262048532e-218
```

The chi-square statistic of 992.856 and a very small p-value of 6.41e-218.

□This suggest that there is a significant association between the variables being tested.

□Therefore, we can reject the null hypothesis of no association and conclude that there is a statistically significant relationship between the variables.

🔶Kruskal-Wallis test

In [23]:

```
from scipy.stats import kruskal

stat, p_val = kruskal(df[df['HeartDisease']=='Yes']['PhysicalHealth'],
                      df[df['HeartDisease']=='No']['PhysicalHealth'])
print(f"Kruskal-Wallis statistic: {stat:.2f}")
print(f"P-value: {p_val:.5f}")
```

```
Kruskal-Wallis statistic: 1961.96
P-value: 0.00000
```

Kruskal-Wallis statistic of 1961.96 and a p-value of 0.00000

□This indicates that there is a significant difference between the groups being compared.

□we can reject the null hypothesis of no difference and conclude that there is a statistically significant difference between the groups based on their ranks.

✓Converting yes and no to 1 and 0

In [24]:

```
df['HeartDisease'] = df['HeartDisease'].map({'Yes': 1, 'No': 0})

# converting Yes and No to 1 and 0
df['Smoking'] = df['Smoking'].map({'Yes': 1, 'No': 0})

# converting Yes and No to 1 and 0
df['AlcoholDrinking'] = df['AlcoholDrinking'].map({'Yes': 1, 'No': 0})

# converting Yes and No to 1 and 0
df['Stroke'] = df['Stroke'].map({'Yes': 1, 'No': 0})

# convert difficulty walking to one-hot encoding
df = pd.concat([df, pd.get_dummies(df['DiffWalking'], prefix='DifficultyWalking')],

# convert sex: {Female, Male} to 1 and 0
df['Sex'] = df['Sex'].map({'Male': 1, 'Female':0})

df = pd.concat([df, pd.get_dummies(df['AgeCategory'], prefix='AgeCategory')], axis=1

# convert ['American Indian/Alaskan Native', 'Asian', 'Black', 'Hispanic', 'Other',
df = pd.concat([df, pd.get_dummies(df['Race'], prefix='Race')], axis=1)

# convert diabetic to one-hot encoding
df = pd.concat([df, pd.get_dummies(df['Diabetic'], prefix='Diabetic')], axis=1)

# converting Yes and No to 1 and 0
df['PhysicalActivity'] = df['PhysicalActivity'].map({'Yes': 1, 'No': 0})

# convert general health from {Excellent, Very good, Good, Fair, Poor} to one-hot en
df = pd.concat([df, pd.get_dummies(df['GenHealth'], prefix='GenHealth')], axis=1)

# converting Yes and No to 1 and 0
df['Asthma'] = df['Asthma'].map({'Yes': 1, 'No': 0})

# converting Yes and No to 1 and 0
df['KidneyDisease'] = df['KidneyDisease'].map({'Yes': 1, 'No': 0})

df['SkinCancer'] = df['SkinCancer'].map({'Yes': 1, 'No': 0})

# drop onehot coded
df = df.drop(['DiffWalking'], axis=1)
df = df.drop(['AgeCategory'], axis=1)
df = df.drop(['Race'], axis=1)
df = df.drop(['GenHealth'], axis=1)
df = df.drop(['Diabetic'], axis=1)
```

In [25]:

```
df
```

Out[25]:

	HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth
0	0	16.60	1	0	0	3.0	30.0
1	0	20.34	0	0	1	0.0	0.0
2	0	26.58	1	0	0	20.0	30.0
3	0	24.21	0	0	0	0.0	0.0
4	0	23.71	0	0	0	28.0	0.0
...	...	...	...	...	...	...	...
79643	0	26.58	1	0	0	0.0	5.0
79644	0	38.65	0	0	0	0.0	10.0
79645	1	29.76	1	0	0	4.0	0.0
79646	0	33.23	1	0	0	0.0	0.0
79647	0	31.75	1	0	0	0.0	27.0

79648 rows × 43 columns

🔶Model Training

In [26]:

```
X = df.drop(['HeartDisease'], axis = 1)
y = df['HeartDisease']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_st
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

Out[26]:

```
((63718, 42), (15930, 42), (63718,), (15930,))
```

In [29]:

```
from sklearn.ensemble import BaggingClassifier

model_bg = BaggingClassifier()
model_bg.fit(X_train,y_train)
print("train score",model_bg.score(X_train,y_train))
print("test score",model_bg.score(X_test,y_test))
```

```
train score 0.9876957845506764
test score 0.9011299435028248
```

In [27]:

```
from sklearn.ensemble import RandomForestClassifier

model_rf = RandomForestClassifier()
model_rf.fit(X_train,y_train)
print("train score",model_rf.score(X_train,y_train))
print("test score",model_rf.score(X_test,y_test))
```

```
train score 0.9991368216202643
test score 0.9063402385436283
```

➖The high training score of the Random Forest Classifier suggests that the model may be overfitting the training data.

➖However, the test score of 0.9063 is still a decent score, indicating that the model can predict the target variable with 90.63% accuracy on unseen data.

In [34]:

```
rfc_reg = RandomForestClassifier(n_estimators=100, max_depth=10, min_samples_split=5
                                  min_samples_leaf=1, max_features='sqrt',
                                  min_impurity_decrease=0.01, random_state=42)

rfc_reg.fit(X_train, y_train)
reg_pred = rfc_reg.predict(X_test)
reg_accuracy = accuracy_score(y_test, reg_pred)
print("Regularized random forest accuracy:", reg_accuracy)
```

```
Regularized random forest accuracy: 0.9141242937853107
```

➖The relatively high accuracy score suggests that the model is capturing the underlying patterns in the data well.

➖The regularization technique used in the model helps to prevent overfitting and improve the model's performance on new data.

In [30]:

```
model_lr = LogisticRegression()
model_lr.fit(X_train,y_train)
print("train score",model_lr.score(X_train,y_train))
print("test score",model_lr.score(X_test,y_test))
```

```
train score 0.9152829655670297
test score 0.9145009416195857
```

➖The relatively high test score indicates that the model can predict the target variable with 91.45%accuracy on unseen data.

➖However, the train score is slightly higher than the test score, indicating that the model may be slightly overfittingthe training data.

➖This means that the model has learned the patterns in the training data too well and may not generalize well to new, unseen data.

In [35]:

```python
from sklearn.metrics import precision_score, recall_score, f1_score

# calculate precision, recall, and F1 score on the test set
rf_pred = rfc_reg.predict(X_test)
precision = precision_score(y_test, rf_pred, average='weighted')
recall = recall_score(y_test, rf_pred, average='weighted')
f1 = f1_score(y_test, rf_pred, average='weighted')

print("Precision: ", precision)
print("Recall: ", recall)
print("F1 score: ", f1)
```

```
Precision:  0.835623224488493
Recall:  0.9141242937853107
F1 score:  0.8731128142530299
```

Precision:-

▬The proportion of predicted positive cases that are actually positive. In this case, the precision score of 0.8356 indicates that the model is quite accurate in predicting positive cases.

▬However, there is still room for improvement in correctly identifying all the positive cases.

Recall:-

▬The proportion of actual positive cases that are correctly identified as positive by the model. The recall score of 0.9141 suggests that the model is able to correctly identify most of the positive cases.

▬But some positive cases are still missed by the model.

F1 score:-

▬The harmonic mean of precision and recall, and it provides a balanced measure of the two metrics.

▬The F1 score of 0.8731 indicates that the model has achieved a reasonable balance between precision and recall.

□Overall, the Regularized Random Forest Classifier is performing well on the given dataset.