IMPORT DATASET

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

LOADING THE CSV FILE (DATASET)

In [2]:
```
df= pd.read_csv('Heart_Attack_prediction_Dataset.csv')
df.head()
```

Out[2]:

	Patient ID	Age	Sex	Cholesterol	Blood Pressure	Heart Rate	Diabetes	Family History	Smoking	Obesity	...	Sedentary Hours Per Day	Income	BI
0	BMW7812	67	Male	208	158/88	72	0	0	1	0	...	6.615001	261404	31.2512
1	CZE1114	21	Male	389	165/93	98	1	1	1	1	...	4.963459	285768	27.1949
2	BNI9906	21	Female	324	174/99	72	1	0	0	0	...	9.463426	235282	28.1765
3	JLN3497	84	Male	383	163/100	73	1	1	1	0	...	7.648981	125640	36.4647
4	GFO8847	66	Male	318	91/88	93	1	1	1	1	...	1.514821	160555	21.8091

5 rows × 26 columns

In [3]:
```
df.columns
```

Out[3]:
```
Index(['Patient ID', 'Age', 'Sex', 'Cholesterol', 'Blood Pressure',
       'Heart Rate', 'Diabetes', 'Family History', 'Smoking', 'Obesity',
       'Alcohol Consumption', 'Exercise Hours Per Week', 'Diet',
       'Previous Heart Problems', 'Medication Use', 'Stress Level',
       'Sedentary Hours Per Day', 'Income', 'BMI', 'Triglycerides',
       'Physical Activity Days Per Week', 'Sleep Hours Per Day', 'Country',
       'Continent', 'Hemisphere', 'Heart Attack Risk'],
      dtype='object')
```

In [4]:
```
df.drop_duplicates(inplace=True)
df.shape
```

Out[4]:
```
(8763, 26)
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8763 entries, 0 to 8762
Data columns (total 26 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   Patient ID                       8763 non-null   object 
 1   Age                              8763 non-null   int64  
 2   Sex                              8763 non-null   object 
 3   Cholesterol                      8763 non-null   int64  
 4   Blood Pressure                   8763 non-null   object 
 5   Heart Rate                       8763 non-null   int64  
 6   Diabetes                         8763 non-null   int64  
 7   Family History                   8763 non-null   int64  
 8   Smoking                          8763 non-null   int64  
 9   Obesity                          8763 non-null   int64  
 10  Alcohol Consumption              8763 non-null   int64  
 11  Exercise Hours Per Week          8763 non-null   float64
 12  Diet                             8763 non-null   object 
 13  Previous Heart Problems          8763 non-null   int64  
 14  Medication Use                   8763 non-null   int64  
 15  Stress Level                     8763 non-null   int64  
 16  Sedentary Hours Per Day          8763 non-null   float64
 17  Income                           8763 non-null   int64  
 18  BMI                              8763 non-null   float64
 19  Triglycerides                    8763 non-null   int64  
 20  Physical Activity Days Per Week  8763 non-null   int64  
 21  Sleep Hours Per Day              8763 non-null   int64  
 22  Country                          8763 non-null   object 
 23  Continent                        8763 non-null   object 
 24  Hemisphere                       8763 non-null   object 
 25  Heart Attack Risk                8763 non-null   int64  
dtypes: float64(3), int64(16), object(7)
memory usage: 1.7+ MB
```

In [6]:
```
df.describe().T
```

Out[6]:

	count	mean	std	min	25%	50%	75%	max
Age	8763.0	53.707977	21.249509	18.000000	35.000000	54.000000	72.000000	90.000000
Cholesterol	8763.0	259.877211	80.863276	120.000000	192.000000	259.000000	330.000000	400.000000
Heart Rate	8763.0	75.021682	20.550948	40.000000	57.000000	75.000000	93.000000	110.000000
Diabetes	8763.0	0.652288	0.476271	0.000000	0.000000	1.000000	1.000000	1.000000
Family History	8763.0	0.492982	0.499979	0.000000	0.000000	0.000000	1.000000	1.000000
Smoking	8763.0	0.896839	0.304186	0.000000	1.000000	1.000000	1.000000	1.000000
Obesity	8763.0	0.501426	0.500026	0.000000	0.000000	1.000000	1.000000	1.000000
Alcohol Consumption	8763.0	0.598083	0.490313	0.000000	0.000000	1.000000	1.000000	1.000000
Exercise Hours Per Week	8763.0	10.014284	5.783745	0.002442	4.981579	10.069559	15.050018	19.998709
Previous Heart Problems	8763.0	0.495835	0.500011	0.000000	0.000000	0.000000	1.000000	1.000000
Medication Use	8763.0	0.498345	0.500026	0.000000	0.000000	0.000000	1.000000	1.000000
Stress Level	8763.0	5.469702	2.859622	1.000000	3.000000	5.000000	8.000000	10.000000
Sedentary Hours Per Day	8763.0	5.993690	3.466359	0.001263	2.998794	5.933622	9.019124	11.999313
Income	8763.0	158263.181901	80575.190806	20062.000000	88310.000000	157866.000000	227749.000000	299954.000000
BMI	8763.0	28.891446	6.319181	18.002337	23.422985	28.768999	34.324594	39.997211
Triglycerides	8763.0	417.677051	223.748137	30.000000	225.500000	417.000000	612.000000	800.000000
Physical Activity Days	8763.0	3.489672	2.282687	0.000000	2.000000	3.000000	5.000000	7.000000

	count	mean	std	min	25%	50%	75%	max
Per Week								
Sleep Hours Per Day	8763.0	7.023508	1.988473	4.000000	5.000000	7.000000	9.000000	10.000000
Heart Attack Risk	8763.0	0.358211	0.479502	0.000000	0.000000	0.000000	1.000000	1.000000

DATA CLEANING

In [7]:
```
columns_to_drop = ['Hemisphere', 'Patient ID', 'Continent']
df.drop(columns_to_drop, axis=1, inplace=True)
df.head()
```

Out[7]:

	Age	Sex	Cholesterol	Blood Pressure	Heart Rate	Diabetes	Family History	Smoking	Obesity	Alcohol Consumption	...	Medication Use	Stress Level	Sede Hou
0	67	Male	208	158/88	72	0	0	1	0	0	...	0	9	6.6
1	21	Male	389	165/93	98	1	1	1	1	1	...	0	1	4.9
2	21	Female	324	174/99	72	1	0	0	0	0	...	1	9	9.4
3	84	Male	383	163/100	73	1	1	1	0	1	...	0	9	7.6
4	66	Male	318	91/88	93	1	1	1	1	0	...	0	6	1.5

5 rows × 23 columns

In [8]:
```
# Split 'Blood Pressure' into systolic/diastolic
if "Blood Pressure" in df.columns:
    bp_split = df["Blood Pressure"].str.split("/", expand=True)
    df["Systolic_BP"] = pd.to_numeric(bp_split[0], errors="coerce")
    df["Diastolic_BP"] = pd.to_numeric(bp_split[1], errors="coerce")
```

```
    df.drop("Blood Pressure", axis=1, inplace=True)

#  Confirm final structure
print("Cleaning, rounding & normalization complete!")
print(f"Final shape: {df.shape}")
print("Data preview:")
print(df.head())
```

```
Cleaning, rounding & normalization complete!
Final shape: (8763, 24)
Data preview:
   Age     Sex  Cholesterol  Heart Rate  Diabetes  Family History  Smoking  \
0   67    Male          208          72         0               0        1
1   21    Male          389          98         1               1        1
2   21  Female          324          72         1               0        0
3   84    Male          383          73         1               1        1
4   66    Male          318          93         1               1        1

   Obesity  Alcohol Consumption  Exercise Hours Per Week  ...  \
0        0                    0                 4.168189  ...
1        1                    1                 1.813242  ...
2        0                    0                 2.078353  ...
3        0                    1                 9.828130  ...
4        1                    0                 5.804299  ...

  Sedentary Hours Per Day  Income        BMI  Triglycerides  \
0                6.615001  261404  31.251233            286
1                4.963459  285768  27.194973            235
2                9.463426  235282  28.176571            587
3                7.648981  125640  36.464704            378
4                1.514821  160555  21.809144            231

   Physical Activity Days Per Week  Sleep Hours Per Day    Country  \
0                                0                    6  Argentina
1                                1                    7     Canada
2                                4                    4     France
3                                3                    4     Canada
4                                1                    5   Thailand

   Heart Attack Risk  Systolic_BP  Diastolic_BP
0                  0          158            88
1                  0          165            93
2                  0          174            99
3                  0          163           100
4                  0           91            88

[5 rows x 24 columns]
```

In [9]:

```
#Check for missing values
missing_summary =df.isnull().sum()
```

```
total_missing = missing_summary.sum()

print("\nMissing values per column:")
print(missing_summary)

print("\nData types summary after cleaning:")
print(df.info())
```

```
Missing values per column:
Age                                0
Sex                                0
Cholesterol                        0
Heart Rate                         0
Diabetes                           0
Family History                     0
Smoking                            0
Obesity                            0
Alcohol Consumption                0
Exercise Hours Per Week            0
Diet                               0
Previous Heart Problems            0
Medication Use                     0
Stress Level                       0
Sedentary Hours Per Day            0
Income                             0
BMI                                0
Triglycerides                      0
Physical Activity Days Per Week    0
Sleep Hours Per Day                0
Country                            0
Heart Attack Risk                  0
Systolic_BP                        0
Diastolic_BP                       0
dtype: int64

Data types summary after cleaning:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8763 entries, 0 to 8762
Data columns (total 24 columns):
 #   Column                           Non-Null Count  Dtype 
---  ------                           --------------  ----- 
 0   Age                              8763 non-null   int64 
 1   Sex                              8763 non-null   object
 2   Cholesterol                      8763 non-null   int64 
 3   Heart Rate                       8763 non-null   int64 
 4   Diabetes                         8763 non-null   int64 
 5   Family History                   8763 non-null   int64 
 6   Smoking                          8763 non-null   int64 
 7   Obesity                          8763 non-null   int64 
 8   Alcohol Consumption              8763 non-null   int64 
```

```
 9   Exercise Hours Per Week          8763 non-null   float64
 10  Diet                             8763 non-null   object 
 11  Previous Heart Problems          8763 non-null   int64  
 12  Medication Use                   8763 non-null   int64  
 13  Stress Level                     8763 non-null   int64  
 14  Sedentary Hours Per Day          8763 non-null   float64
 15  Income                           8763 non-null   int64  
 16  BMI                              8763 non-null   float64
 17  Triglycerides                    8763 non-null   int64  
 18  Physical Activity Days Per Week  8763 non-null   int64  
 19  Sleep Hours Per Day              8763 non-null   int64  
 20  Country                          8763 non-null   object 
 21  Heart Attack Risk                8763 non-null   int64  
 22  Systolic_BP                      8763 non-null   int64  
 23  Diastolic_BP                     8763 non-null   int64  
dtypes: float64(3), int64(18), object(3)
memory usage: 1.6+ MB
None
```

FEATURE ENGINEERING

In [10]:

```
# Feature Engineering
df["BMI_Stress"] = df["BMI"] * df["Stress Level"]
df["Activity_Ratio"] = df["Exercise Hours Per Week"] / (df["Sedentary Hours Per Day"] + 1)
df["BP_Product"] = df["Systolic_BP"] * df["Diastolic_BP"]
df["Sleep_Stress_Interaction"] = df["Sleep Hours Per Day"] * df["Stress Level"]
df["Substance_Use"] = df["Smoking"] * df["Alcohol Consumption"]
```

In [11]:

```
print("\nData types summary after cleaning:")
print(df.info())
```

```
Data types summary after cleaning:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8763 entries, 0 to 8762
Data columns (total 29 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   Age                              8763 non-null   int64
 1   Sex                              8763 non-null   object
 2   Cholesterol                      8763 non-null   int64
 3   Heart Rate                       8763 non-null   int64
 4   Diabetes                         8763 non-null   int64
 5   Family History                   8763 non-null   int64
 6   Smoking                          8763 non-null   int64
 7   Obesity                          8763 non-null   int64
 8   Alcohol Consumption              8763 non-null   int64
 9   Exercise Hours Per Week          8763 non-null   float64
 10  Diet                             8763 non-null   object
 11  Previous Heart Problems          8763 non-null   int64
 12  Medication Use                   8763 non-null   int64
 13  Stress Level                     8763 non-null   int64
 14  Sedentary Hours Per Day          8763 non-null   float64
 15  Income                           8763 non-null   int64
 16  BMI                              8763 non-null   float64
 17  Triglycerides                    8763 non-null   int64
 18  Physical Activity Days Per Week  8763 non-null   int64
 19  Sleep Hours Per Day              8763 non-null   int64
 20  Country                          8763 non-null   object
 21  Heart Attack Risk                8763 non-null   int64
 22  Systolic_BP                      8763 non-null   int64
 23  Diastolic_BP                     8763 non-null   int64
 24  BMI_Stress                       8763 non-null   float64
 25  Activity_Ratio                   8763 non-null   float64
 26  BP_Product                       8763 non-null   int64
 27  Sleep_Stress_Interaction         8763 non-null   int64
 28  Substance_Use                    8763 non-null   int64
dtypes: float64(5), int64(21), object(3)
memory usage: 1.9+ MB
None
```

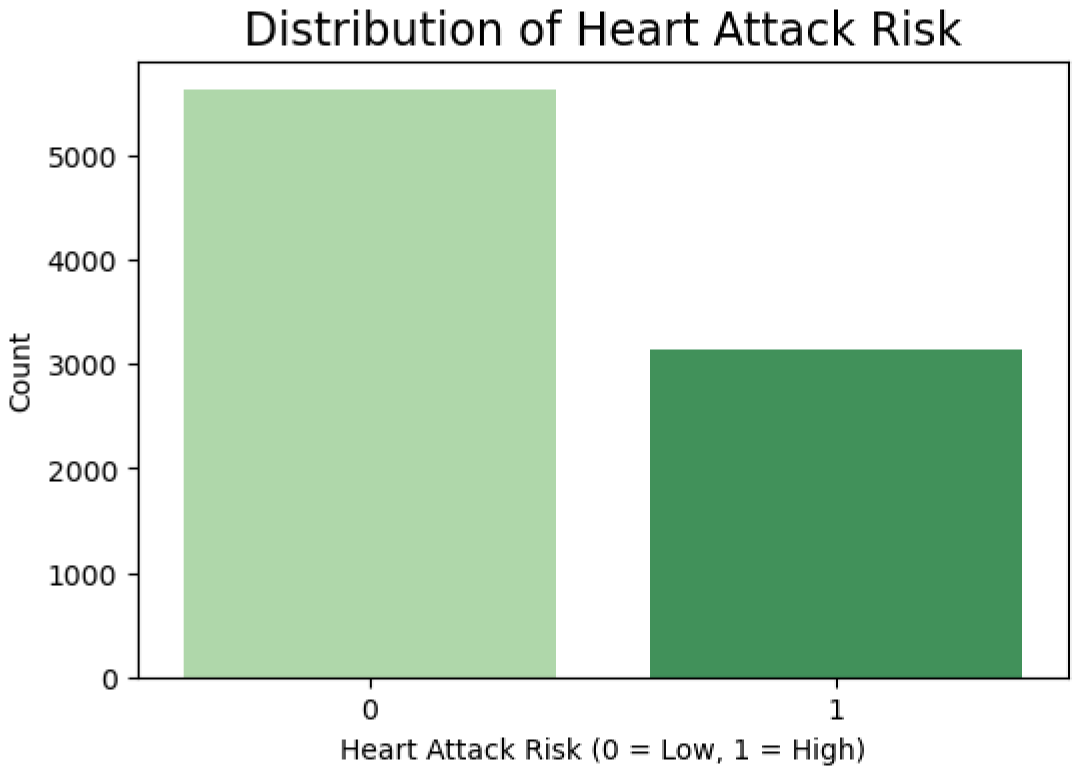
In [12]:
```
df.nunique()
```

Out[12]:
```
Age                               73
Sex                                2
Cholesterol                      281
Heart Rate                        71
Diabetes                           2
Family History                     2
Smoking                            2
Obesity                            2
Alcohol Consumption                2
Exercise Hours Per Week         8763
Diet                               3
Previous Heart Problems            2
Medication Use                     2
Stress Level                      10
Sedentary Hours Per Day         8763
Income                          8615
BMI                             8763
Triglycerides                    771
Physical Activity Days Per Week    8
Sleep Hours Per Day                7
Country                           20
Heart Attack Risk                  2
Systolic_BP                       91
Diastolic_BP                      51
BMI_Stress                      8763
Activity_Ratio                  8763
BP_Product                      2835
Sleep_Stress_Interaction          39
Substance_Use                      2
dtype: int64
```

EXPLORATORY DATA ANALYSIS AND VISUALIZINGS

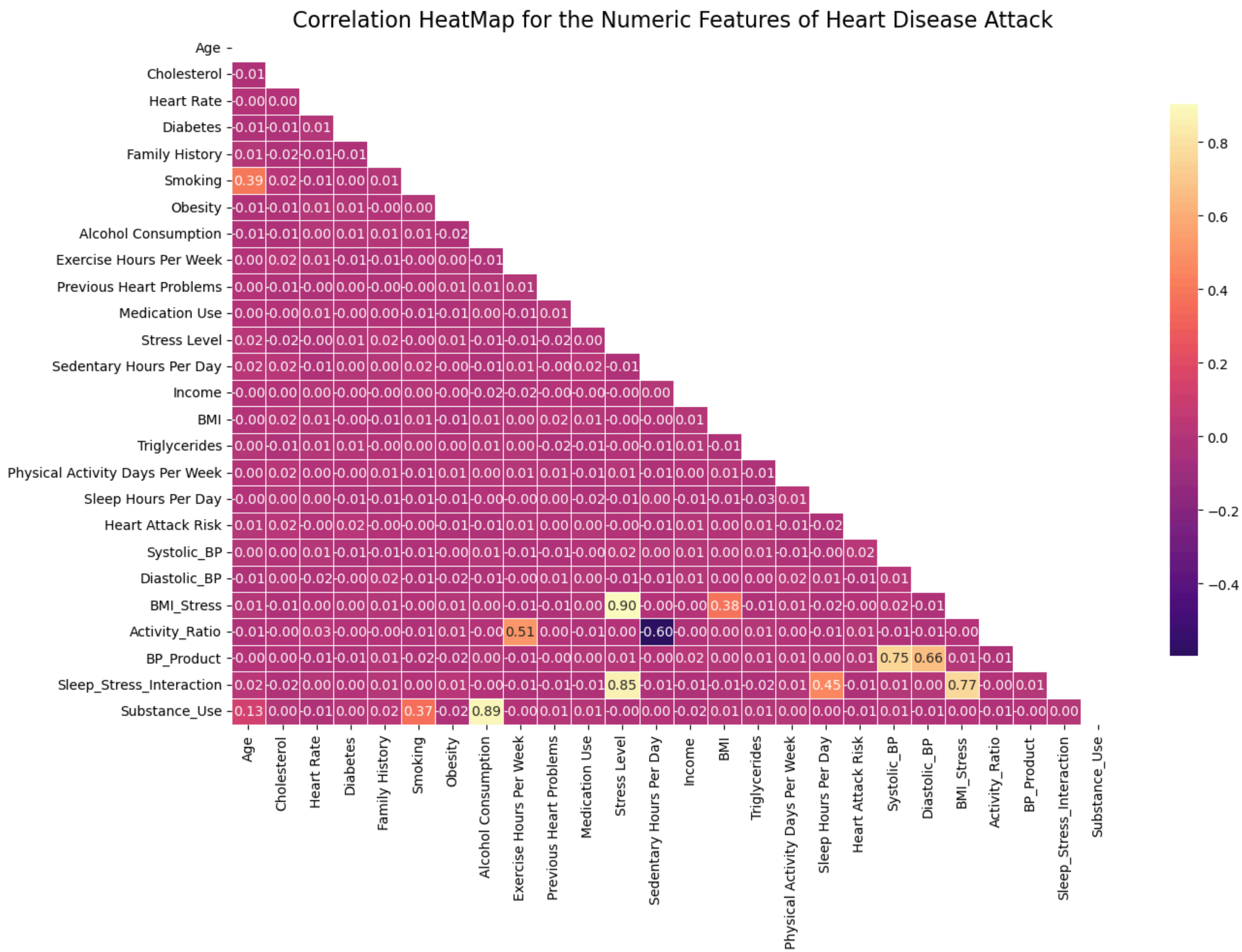
In [13]:
```
# ---  Distribution of Target Variable --
plt.figure(figsize=(6,4))
sns.countplot(x="Heart Attack Risk", hue="Heart Attack Risk",data=df, palette="Greens",
legend=False
)
plt.title("Distribution of Heart Attack Risk", fontsize = 16)
plt.xlabel("Heart Attack Risk (0 = Low, 1 = High)")
```

```
plt.ylabel("Count")
plt.show()
```



In [14]:

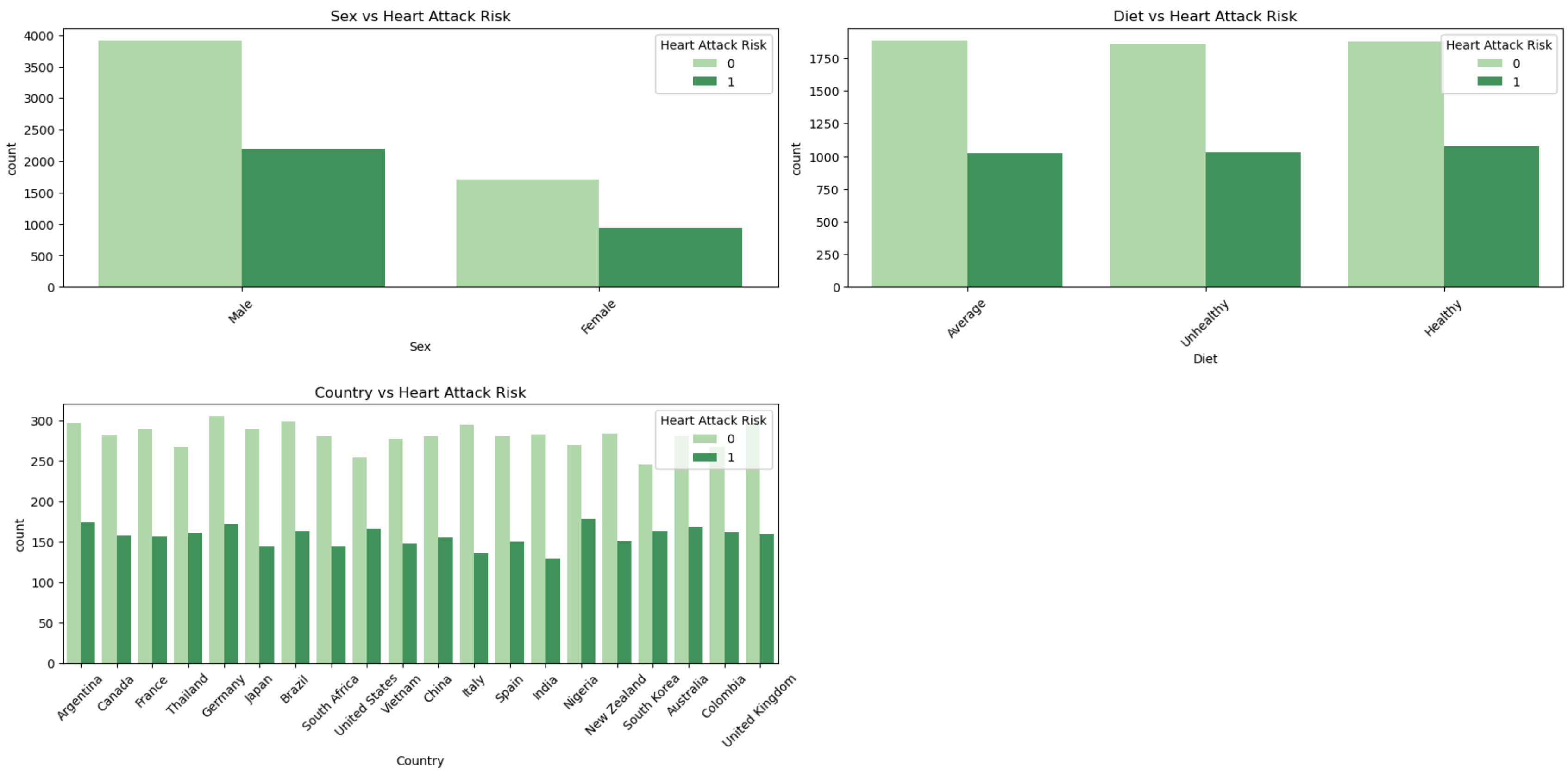
```
# ---  Correlation Heatmap --
plt.figure(figsize=(14,10))
 # Select only numeric columns for correlation
corr = df.select_dtypes(include=['number']).corr()
 # Focus on stronger correlations only
mask = np.triu(np.ones_like(corr, dtype=bool))  # mask upper triangle
sns.heatmap(corr, mask=mask, cmap="magma", center=0, annot=True,fmt=".2f",linewidths=0.5,
cbar_kws={"shrink": 0.8}
)
plt.title("Correlation HeatMap for the Numeric Features of Heart Disease Attack", fontsize=16)
plt.tight_layout()
plt.show()
```



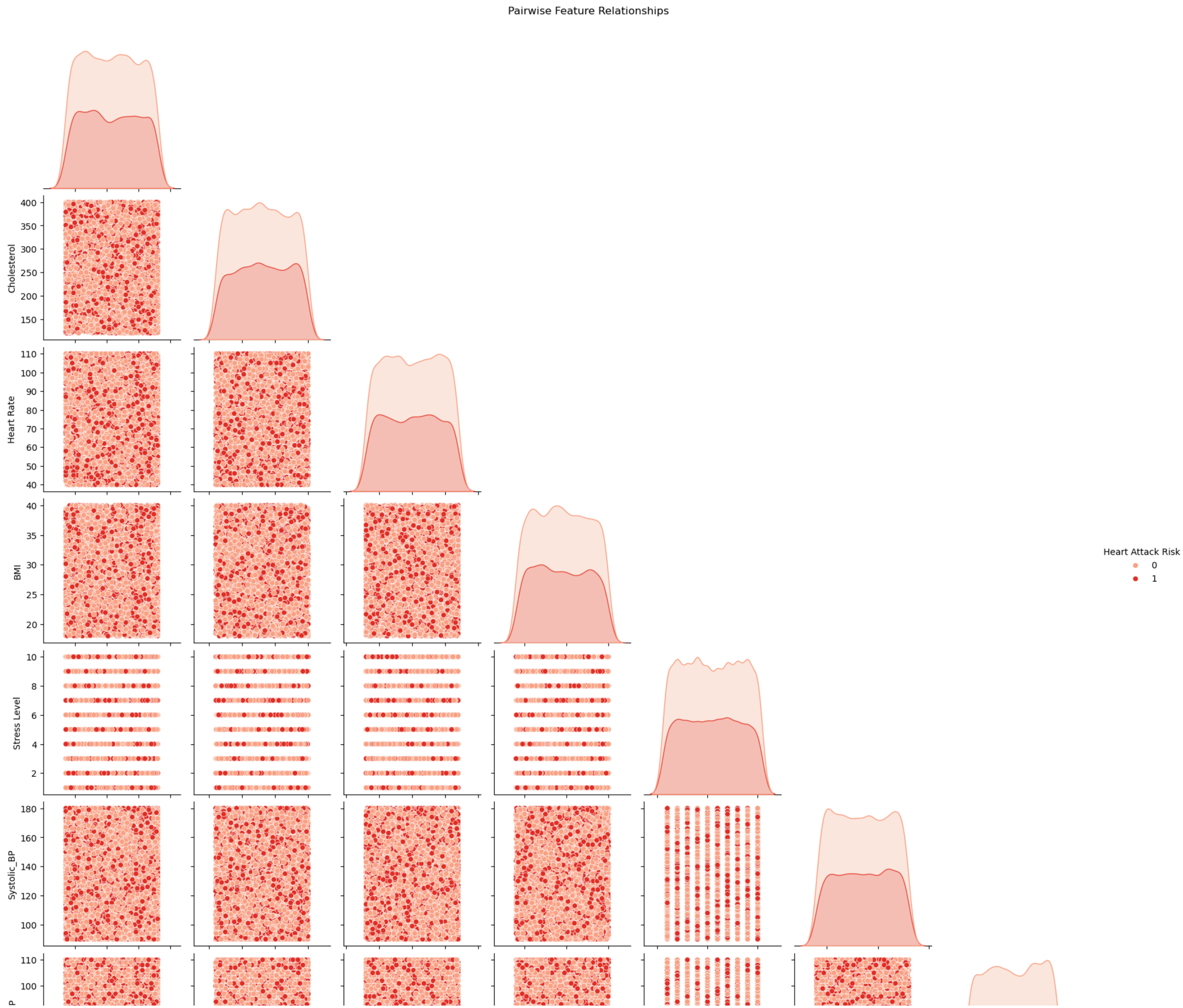
In [15]:

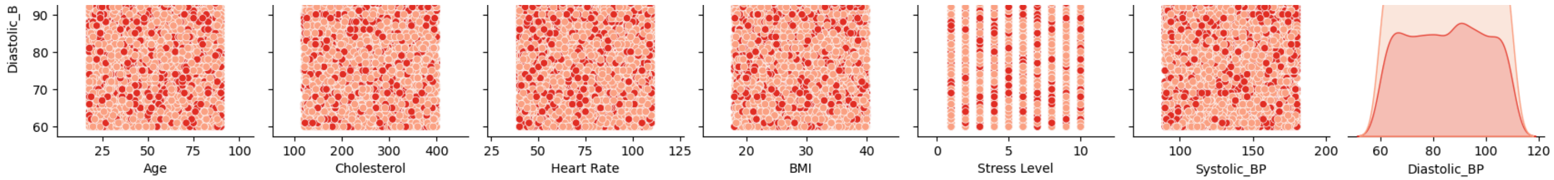
```python
# Target Distribution by Category
cat_cols = ['Sex', 'Diet', 'Country']
```

```
plt.figure(figsize=(18,12))
for i, col in enumerate(cat_cols, 1):
    plt.subplot(3, 2, i)
    sns.countplot(data=df, x=col, hue="Heart Attack Risk", palette="Greens")
    plt.title(f"{col} vs Heart Attack Risk")
    plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



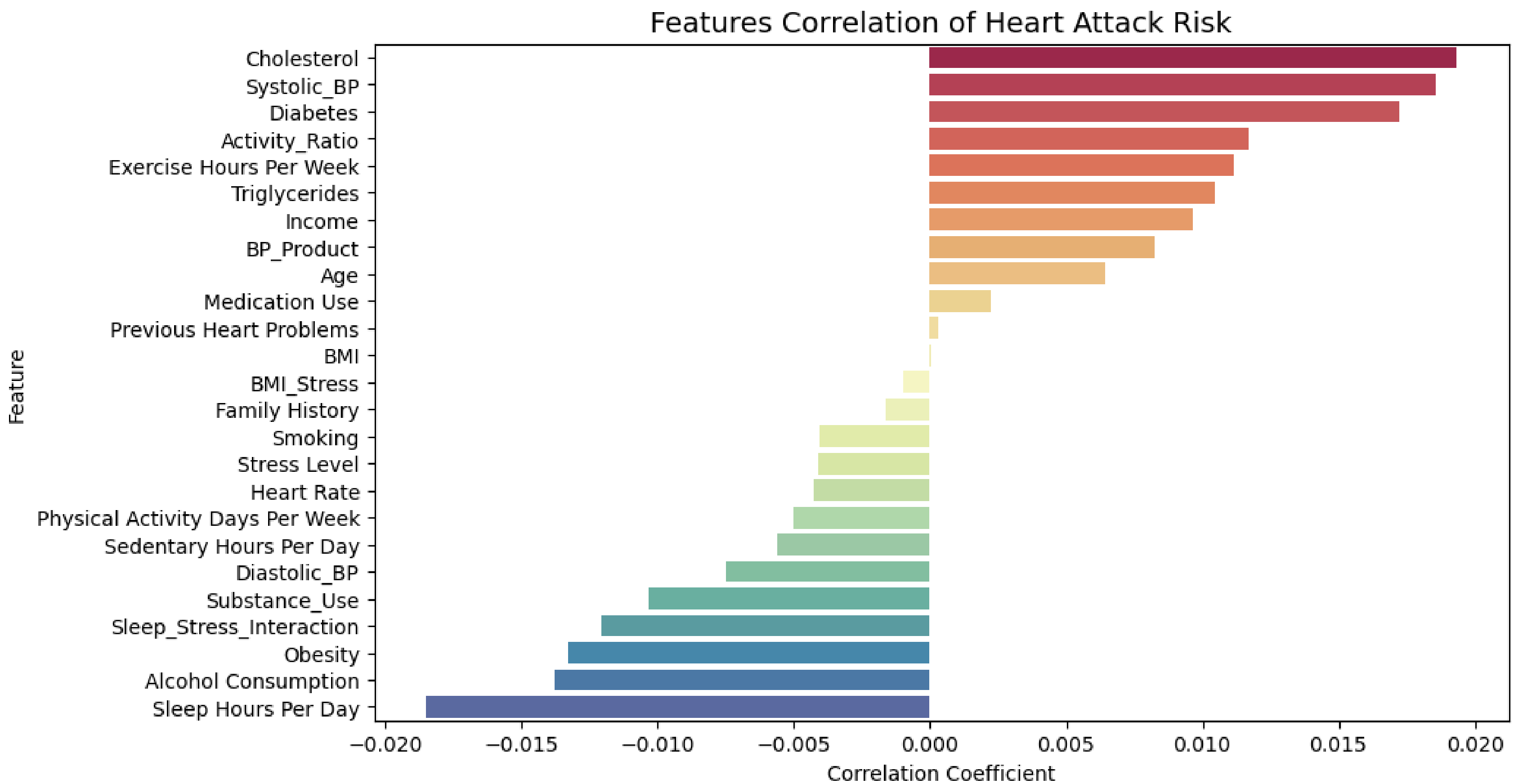
In [16]:
```
# Pairwise Relationships (Hidden Correlations)
sns.pairplot( df[['Age', 'Cholesterol', 'Heart Rate', 'BMI', 'Stress Level', 'Systolic_BP', 'Diastolic_BP', 'Heart At
hue="Heart Attack Risk", palette="Reds", diag_kind="kde", corner=True
)
plt.suptitle("Pairwise Feature Relationships", y=1.02)
plt.show()
```


Pairwise Feature Relationships
Cholesterol
Heart Rate
BMI
Stress Level
Systolic_BP
Heart Attack Risk
0
1



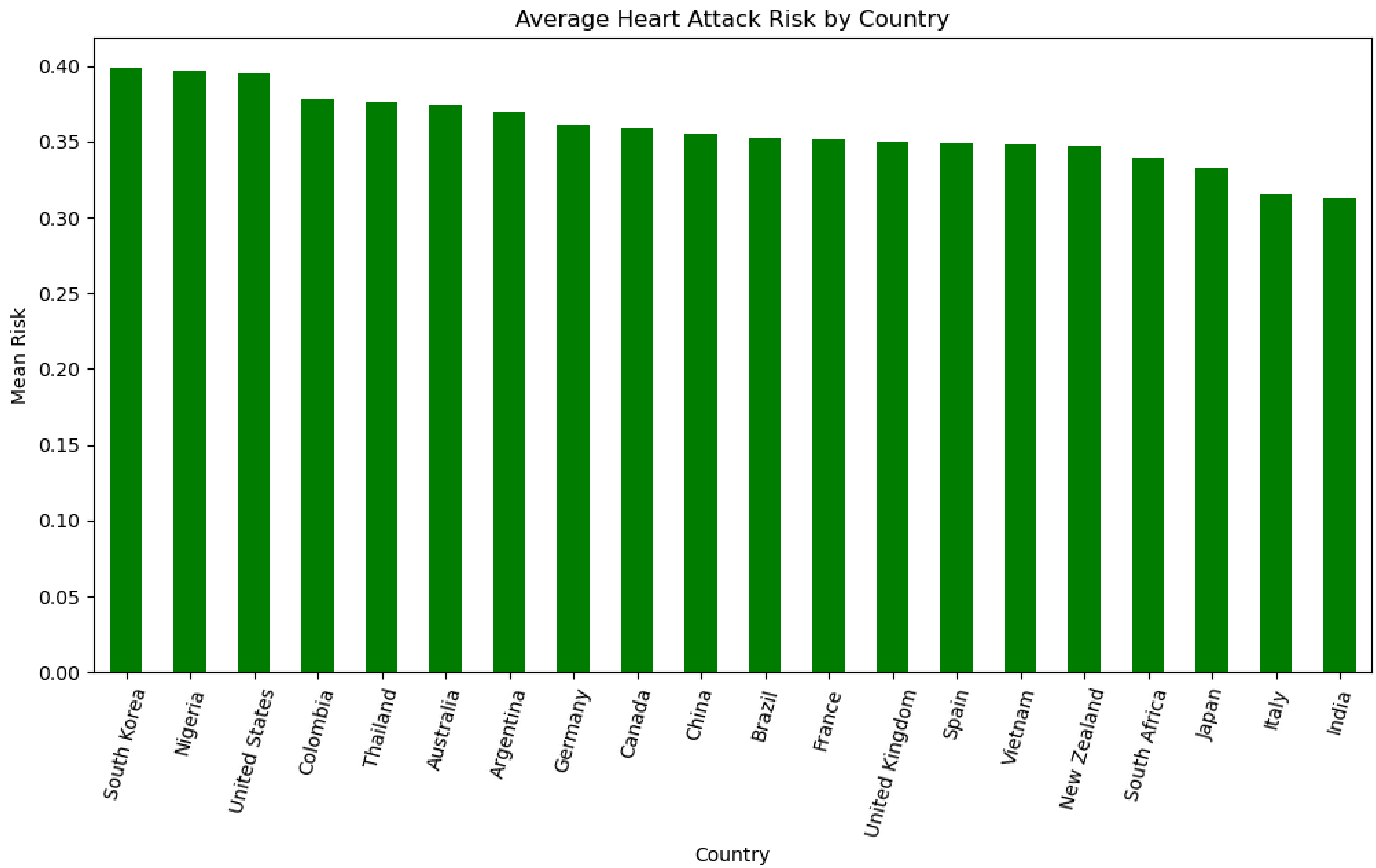
In [17]:

```
# Sorted Correlation with Target
numeric_df = df.select_dtypes(include=['number'])
target_corr = (numeric_df.corr()['Heart Attack Risk']
 .drop('Heart Attack Risk')
 .sort_values(ascending=False)
 )
 # --- Visualize correlation strength --
plt.figure(figsize=(10, 6))
sns.barplot(x=target_corr, y=target_corr.index, palette="Spectral")
plt.title("Features Correlation of Heart Attack Risk", fontsize=14)
plt.xlabel("Correlation Coefficient")
plt.ylabel("Feature")
plt.show()
```

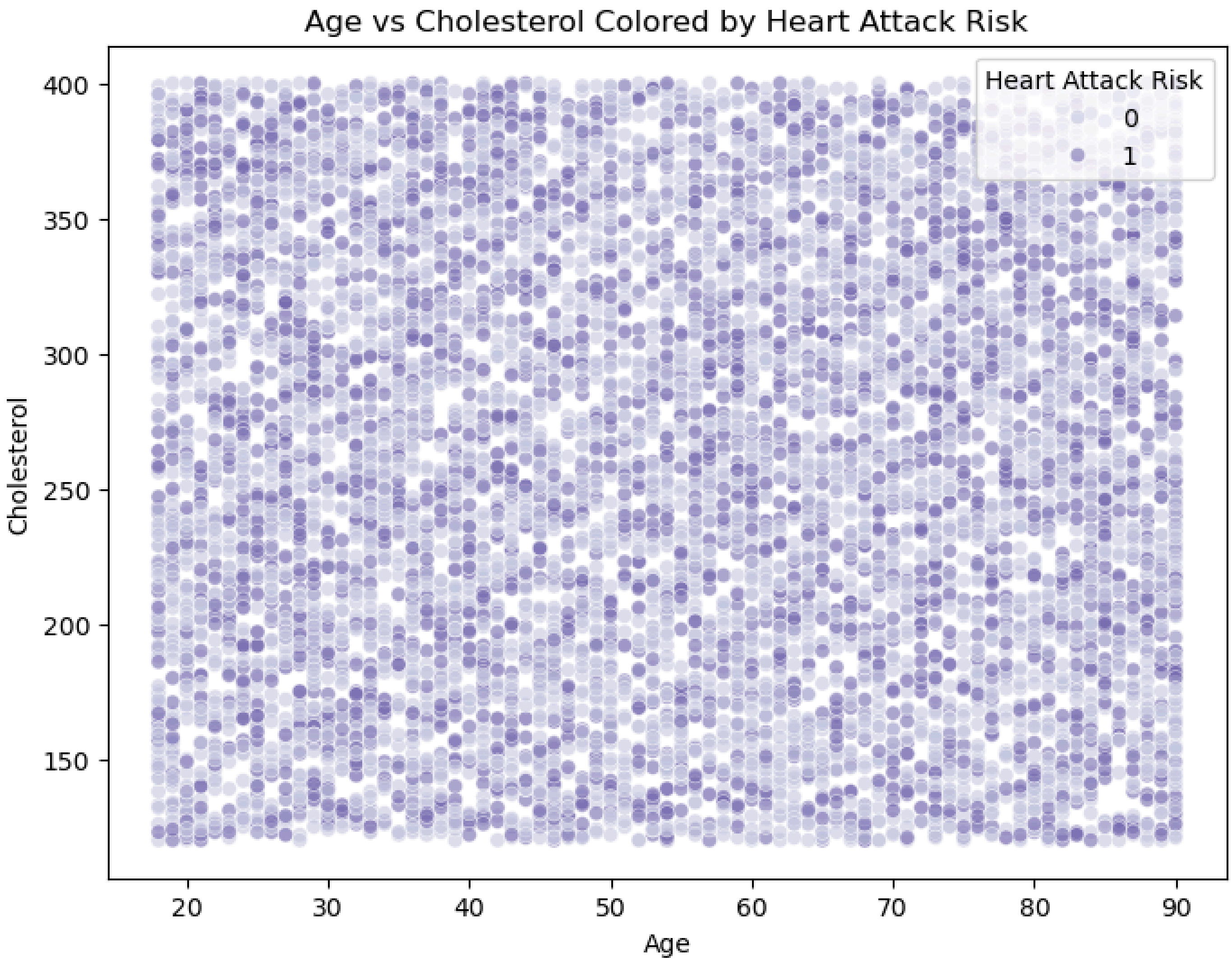


In [18]:

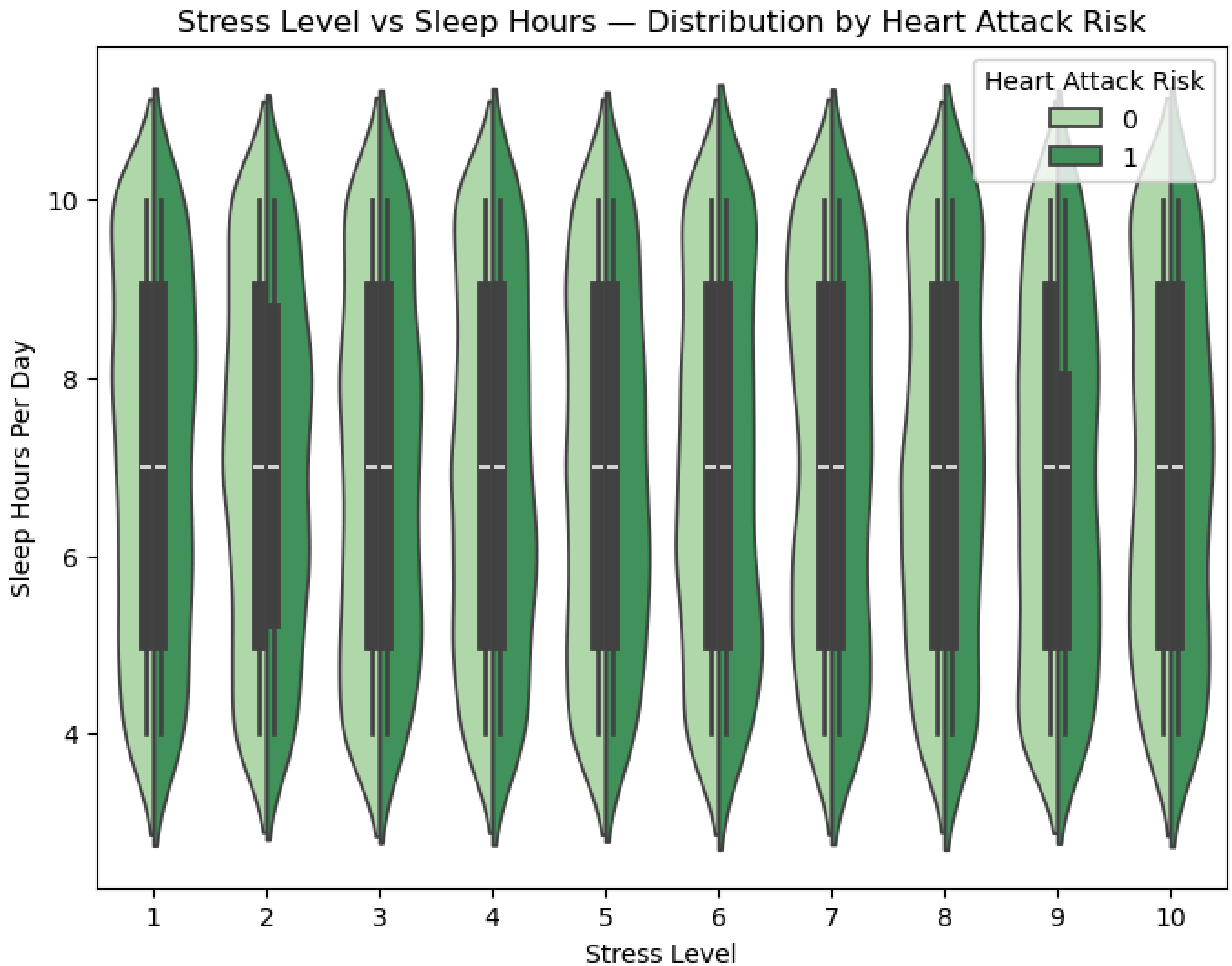
```
# Geographic Patterns(country)
risk_by_country = df.groupby('Country')['Heart Attack Risk'].mean().sort_values(ascending=False)
plt.figure(figsize=(12,6))
risk_by_country.plot(kind='bar', color='green')
plt.title("Average Heart Attack Risk by Country")
plt.ylabel("Mean Risk")
plt.xticks(rotation=75)
plt.show()
```



In [19]:
```
# Age vs Cholesterol Colored by Heart Attack Risk
plt.figure(figsize=(8,6))
sns.scatterplot(data= df, x='Age', y='Cholesterol', hue='Heart Attack Risk', palette='Purples', alpha=0.6
)
plt.title("Age vs Cholesterol Colored by Heart Attack Risk")
plt.show()
```

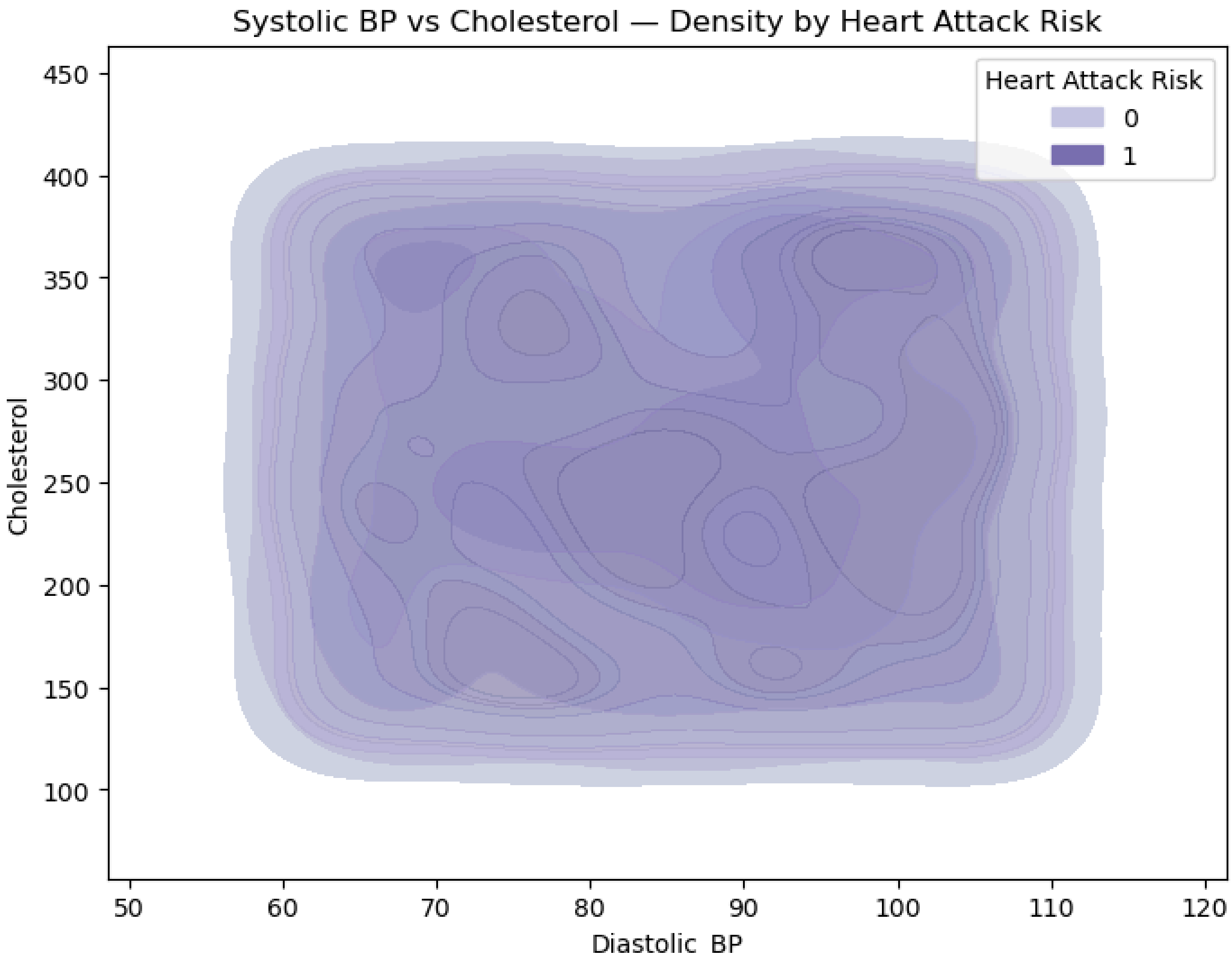


In [20]:
```
# Stress Level vs Sleep Hours
plt.figure(figsize=(8,6))
sns.violinplot(data=df, x="Stress Level", y="Sleep Hours Per Day", hue="Heart Attack Risk", split=True, palette="Gree
)
plt.title("Stress Level vs Sleep Hours — Distribution by Heart Attack Risk")
plt.show()
```



In [21]:

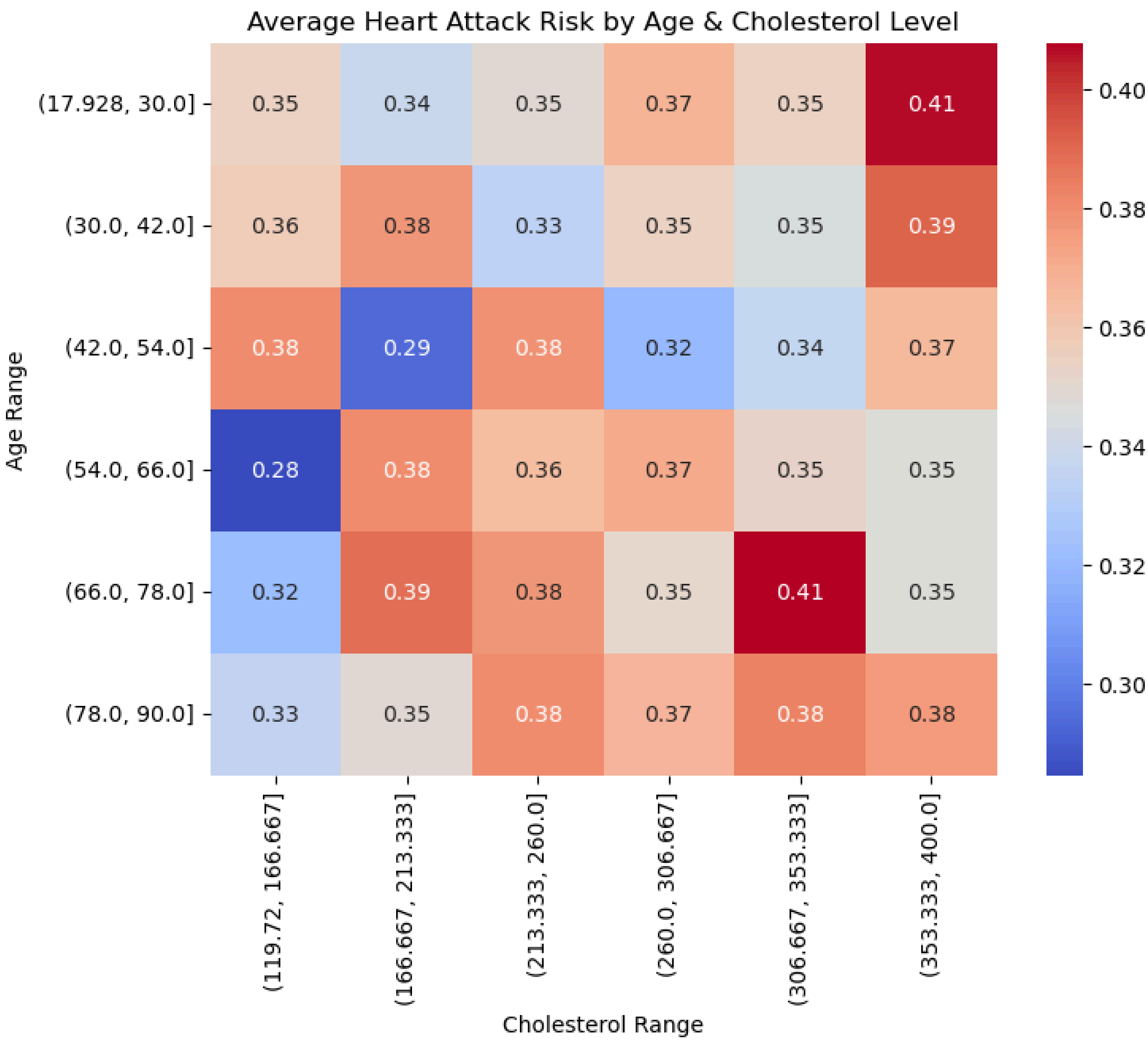
```
# Systolic BP vs Cholesterol
plt.figure(figsize=(8,6))
sns.kdeplot(data=df, x="Diastolic_BP", y="Cholesterol", hue="Heart Attack Risk", fill=True, thresh=0.05, alpha=0.6, 
)
plt.title("Systolic BP vs Cholesterol – Density by Heart Attack Risk")
plt.show()
```



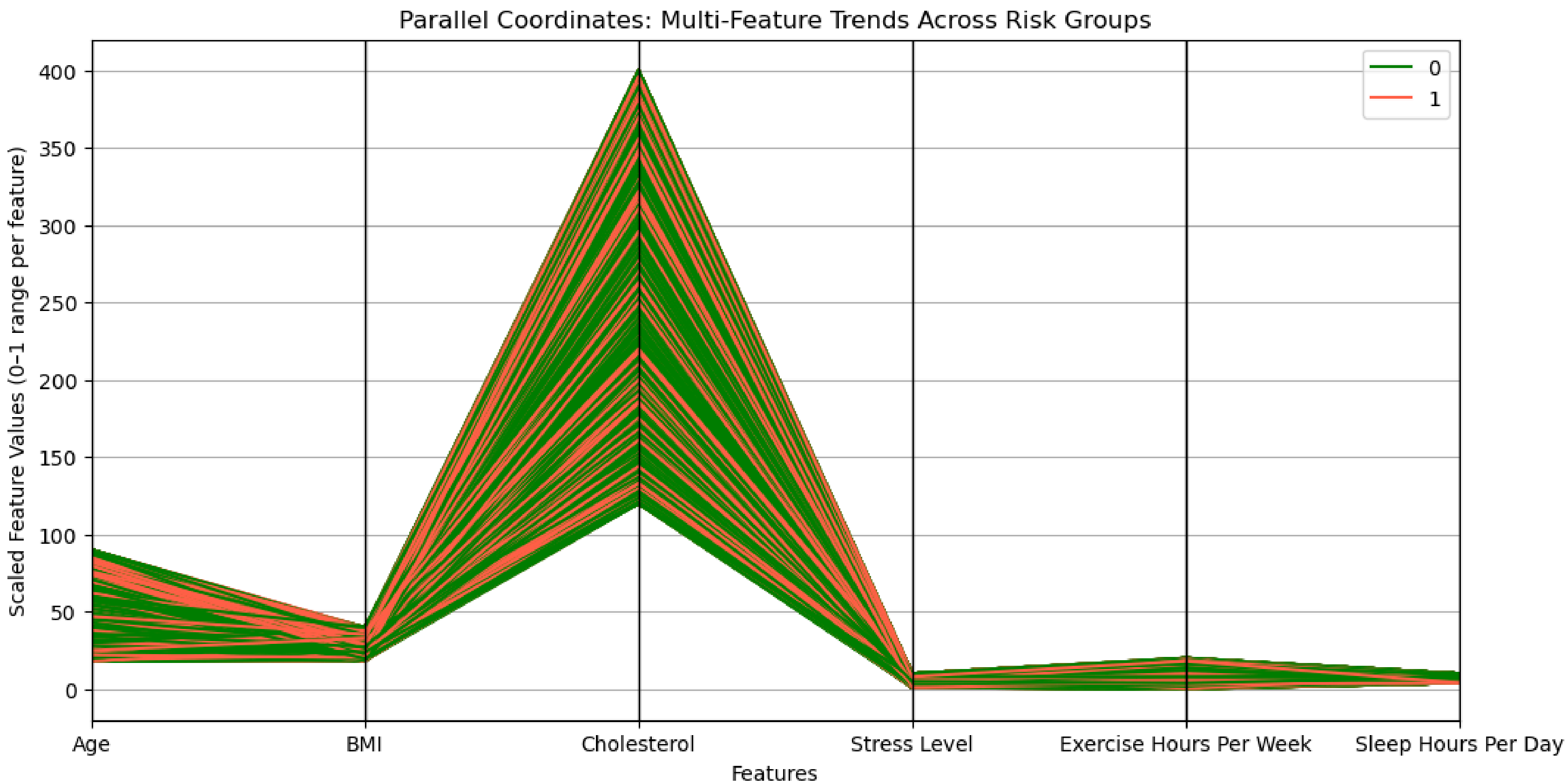
In [22]:

```
# Average Heart Attack Risk by Age & Cholesterol Level
plt.figure(figsize=(8,6))
pivot = df.pivot_table(values="Heart Attack Risk",
index=pd.cut(df["Age"], bins=6),
columns=pd.cut(df["Cholesterol"], bins=6),
aggfunc="mean")
sns.heatmap(pivot, cmap="coolwarm", annot=True, fmt=".2f")
plt.title("Average Heart Attack Risk by Age & Cholesterol Level")
plt.xlabel("Cholesterol Range")
```

```
plt.ylabel("Age Range")
plt.show()
```

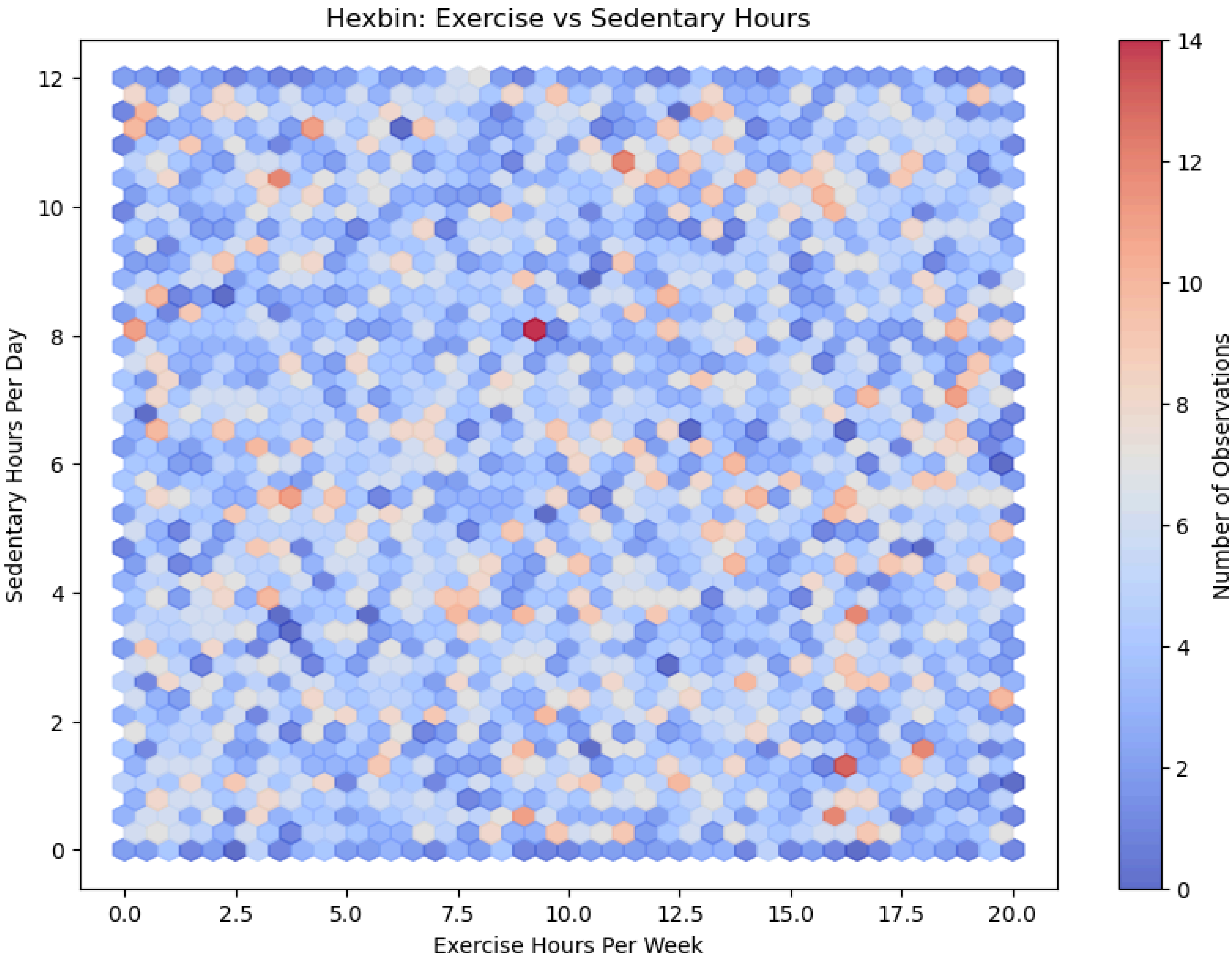


In [23]:
```
# Multi-Feature Trends Across Risk Groups
from pandas.plotting import parallel_coordinates
selected_cols = ["Age", "BMI", "Cholesterol", "Stress Level", "Exercise Hours Per Week", "Sleep Hours Per Day", "Hea
plt.figure(figsize=(12,6))
parallel_coordinates(df[selected_cols], "Heart Attack Risk", color=["green", "tomato"])
plt.title("Parallel Coordinates: Multi-Feature Trends Across Risk Groups")
plt.ylabel("Scaled Feature Values (0-1 range per feature)")
plt.xlabel("Features")
plt.show()
```



In [24]:
```
# Hexbin: Exercise vs Sedentary Hours
plt.figure(figsize=(10,7))
plt.hexbin(df["Exercise Hours Per Week"], df["Sedentary Hours Per Day"], gridsize=40, cmap="coolwarm", alpha=0.8
)
plt.colorbar(label="Number of Observations")
```

```
plt.title("Hexbin: Exercise vs Sedentary Hours")
plt.xlabel("Exercise Hours Per Week")
plt.ylabel("Sedentary Hours Per Day")
plt.show()
```



APPLY SMOTE AND VISUALIZE TARGET BALANCE

BEFORE SMOTE - CHECK CLASS DISTRIBUTION

In [25]:

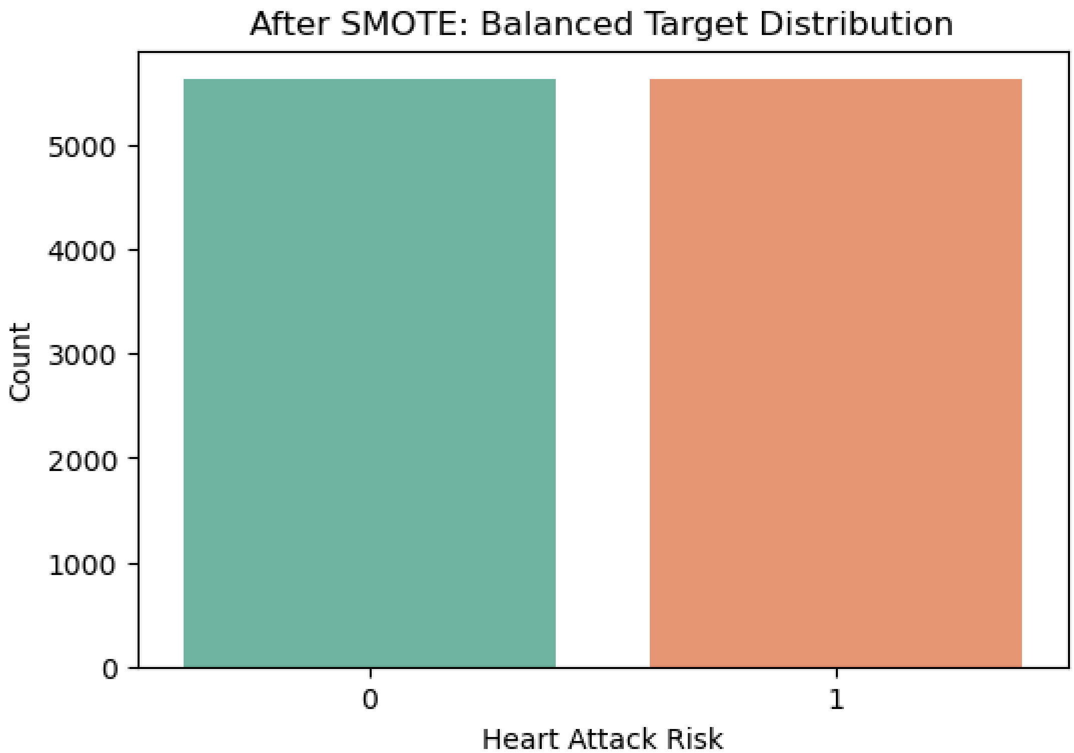
```
from imblearn.over_sampling import SMOTE
from collections import Counter
X = df.drop(columns=["Heart Attack Risk"])
y = df["Heart Attack Risk"]
X = pd.get_dummies(X, drop_first=True)
print("Original class distribution:")
print(Counter(y))
plt.figure(figsize=(6,4))
sns.countplot(x=y, palette="pastel")
plt.title("Before SMOTE: Target Distribution")
plt.xlabel("Heart Attack Risk")
plt.ylabel("Count")
plt.show()
```

```
Original class distribution:
Counter({0: 5624, 1: 3139})
```

AFTER SMOTE — CHECK NEW BALANCE

In [26]:
```
#  Apply SMOTE
smote = SMOTE(random_state=42)
X_res, y_res = smote.fit_resample(X, y)
print("\n After SMOTE class distribution:")
print(Counter(y_res))
plt.figure(figsize=(6,4))
sns.countplot(x=y_res, palette="Set2")
plt.title("After SMOTE: Balanced Target Distribution")
plt.xlabel("Heart Attack Risk")
plt.ylabel("Count")
plt.show()
```

```
 After SMOTE class distribution:
Counter({0: 5624, 1: 5624})
```



CREATE A NEW BALANCED DATAFRAME

In [27]:
```
df_balanced = pd.concat([pd.DataFrame(X_res), pd.Series(y_res, name="Heart Attack Risk")], axis=1)
df_balanced
```

Out[27]:

	Age	Cholesterol	Heart Rate	Diabetes	Family History	Smoking	Obesity	Alcohol Consumption	Exercise Hours Per Week	Previous Heart Problems	...	Country_New Zealand	C
0	67	208	72	0	0	1	0	0	4.168189	0	...	False	
1	21	389	98	1	1	1	1	1	1.813242	1	...	False	
2	21	324	72	1	0	0	0	0	2.078353	1	...	False	
3	84	383	73	1	1	1	0	1	9.828130	1	...	False	
4	66	318	93	1	1	1	1	0	5.804299	1	...	False	
...	...	...	...	...	...	...	...	...	...	...	...	...	
11243	43	201	100	1	0	1	0	0	2.202700	0	...	False	
11244	65	303	74	0	0	1	1	0	10.382115	1	...	False	
11245	25	363	62	1	0	1	1	0	9.892264	0	...	False	
11246	24	293	96	0	0	0	0	0	11.497022	1	...	False	
11247	60	312	58	1	0	1	0	1	9.960904	0	...	False	

11248 rows × 48 columns

PERFORMING FEATURE SELECTION USING DIFFERENT METHODS FOR SELECTION ON THE NEW DATAFRAME AND TRAINING DIFFERENT MODELS WITH THE SELECTED FEATURES WITH HYPERPARAMETER TURNING

IMPORTING DEPENDENCIES

In [28]:
```
# Imports
from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import SelectKBest, chi2, RFE, mutual_info_classif
from sklearn.linear_model import LogisticRegression, Lasso
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from catboost import CatBoostClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

PREPARING DATA FOR NORMALIZING AND TRAINING

In [29]:
```
#  Split data
X = df_balanced.drop("Heart Attack Risk", axis=1)
y = df_balanced["Heart Attack Risk"]

scaler = MinMaxScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, stratify=y, random_state=42)
```

MODELS DEFINATIONN AND PARAMETERS FOR TURNING

In [30]:
```
models_params = {
    "LogisticRegression": (LogisticRegression(max_iter=2000), {
        "C": [0.01, 0.1, 1, 10],
        "solver": ["liblinear", "lbfgs"]
    }),
    "DecisionTreeClassifier": (DecisionTreeClassifier(random_state=42), {
        "max_depth": [3, 5, 10, None],
        "min_samples_split": [2, 5, 10],
        "criterion":['gini','entropy']
    }),
    "XGBoost": (XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42), {
        "n_estimators": [100, 200],
        "learning_rate": [0.01, 0.1, 0.2],
        "max_depth": [3, 5, 7]
```

```
    }),
    "CatBoost": (CatBoostClassifier(verbose=0, random_state=42), {
        "iterations": [100, 200],
        "learning_rate": [0.01, 0.1, 0.2],
        "depth": [3, 5, 7]
    }),
    "RandomForest": (RandomForestClassifier(random_state=42), {
       "n_estimators": [100, 200],
        "min_samples_split": [2, 5],
        "max_depth": [5, 10, 15]
    })
}
```

DEFINING FEATURE SELECTION METHODS

In [31]:
```
feature_selectors = {}
# 1. Univariate (SelectKBest)
skb = SelectKBest(score_func=chi2, k=10)
skb.fit(X_scaled, y)
feature_selectors['SelectKBest'] = X_scaled.columns[skb.get_support()]

# 2. RFE with Logistic Regression
rfe = RFE(LogisticRegression(max_iter=1000), n_features_to_select=10)
rfe.fit(X_scaled, y)
feature_selectors['RFE'] = X_scaled.columns[rfe.support_]

# 3. Random Forest Importance
rf = RandomForestClassifier(random_state=42)
rf.fit(X_scaled, y)
rf_features = X_scaled.columns[np.argsort(rf.feature_importances_)[-10:]]
feature_selectors['RandomForest_Importance'] = rf_features

# 4. L1 Regularization (Lasso)
lasso = Lasso(alpha=0.001, max_iter=10000)
lasso.fit(X_scaled, y)
lasso_features = X_scaled.columns[np.abs(lasso.coef_) > 1e-4]
feature_selectors['L1_Lasso'] = lasso_features

# 5. Mutual Information
mi = mutual_info_classif(X_scaled, y)
```

```
mi_features = X_scaled.columns[np.argsort(mi)[-10:]]
feature_selectors['Mutual_Info'] = mi_features
```

TRAINING AND EVALUATING MODELS

In [32]:
```
results = []
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
for method, features in feature_selectors.items():
    print(f"\n Feature Selection: {method}")
    X_train_fs, X_test_fs = X_train[features], X_test[features]

    for name, (model, params) in models_params.items():
        print(f"Training {name} ...")
        grid = GridSearchCV(model, params, cv=cv, scoring='accuracy', n_jobs=-1)
        grid.fit(X_train_fs, y_train)
        best_model = grid.best_estimator_
        y_pred = best_model.predict(X_test_fs)
        acc = accuracy_score(y_test, y_pred)
        pre = precision_score(y_test, y_pred)
        recall = recall_score(y_test, y_pred)
        f1     = f1_score(y_test, y_pred)

        results.append({
            "Feature_Selection": method,
            "Model": name,
            "Best_Params": grid.best_params_,
            "Accuracy": acc,
            "Precision": pre,
            "Recall": recall,
            "F1_Score": f1
        })
```

```
 Feature Selection: SelectKBest
Training LogisticRegression ...
Training DecisionTreeClassifier ...
Training XGBoost ...
Training CatBoost ...
Training RandomForest ...

 Feature Selection: RFE
Training LogisticRegression ...
Training DecisionTreeClassifier ...
Training XGBoost ...
Training CatBoost ...
Training RandomForest ...

 Feature Selection: RandomForest_Importance
Training LogisticRegression ...
Training DecisionTreeClassifier ...
Training XGBoost ...
Training CatBoost ...
Training RandomForest ...

 Feature Selection: L1_Lasso
Training LogisticRegression ...
Training DecisionTreeClassifier ...
Training XGBoost ...
Training CatBoost ...
Training RandomForest ...

 Feature Selection: Mutual_Info
Training LogisticRegression ...
Training DecisionTreeClassifier ...
Training XGBoost ...
Training CatBoost ...
Training RandomForest ...
```

COMPARING RESULTS

In [33]:
```
results_df = pd.DataFrame(results)
results_df = results_df.sort_values(by="Accuracy", ascending=False).reset_index(drop=True)
```

```
print("\n Model Performance Summary:")
results_df
```

```
Model Performance Summary:
```

Out[33]:

	Feature_Selection	Model	Best_Params	Accuracy	Precision	Recall	F1_Score
0	L1_Lasso	LogisticRegression	{'C': 10, 'solver': 'liblinear'}	0.692889	0.800554	0.513778	0.625880
1	L1_Lasso	RandomForest	{'max_depth': 15, 'min_samples_split': 2, 'n_e...	0.684889	0.729581	0.587556	0.650911
2	L1_Lasso	XGBoost	{'learning_rate': 0.1, 'max_depth': 3, 'n_esti...	0.678222	0.714898	0.592889	0.648202
3	L1_Lasso	CatBoost	{'depth': 5, 'iterations': 100, 'learning_rate...	0.665333	0.706667	0.565333	0.628148
4	Mutual_Info	XGBoost	{'learning_rate': 0.2, 'max_depth': 7, 'n_esti...	0.644889	0.647378	0.636444	0.641865
5	RandomForest_Importance	CatBoost	{'depth': 7, 'iterations': 200, 'learning_rate...	0.643556	0.632486	0.685333	0.657850
6	Mutual_Info	CatBoost	{'depth': 5, 'iterations': 200, 'learning_rate...	0.643111	0.657227	0.598222	0.626338
7	SelectKBest	CatBoost	{'depth': 5, 'iterations': 100, 'learning_rate...	0.640889	0.679502	0.533333	0.597610
8	SelectKBest	RandomForest	{'max_depth': 5, 'min_samples_split': 2, 'n_es...	0.634222	0.650398	0.580444	0.613434
9	SelectKBest	XGBoost	{'learning_rate': 0.1, 'max_depth': 5, 'n_esti...	0.631556	0.671296	0.515556	0.583208
10	SelectKBest	DecisionTreeClassifier	{'criterion': 'entropy', 'max_depth': 10, 'min...	0.628889	0.676829	0.493333	0.570694
11	RandomForest_Importance	RandomForest	{'max_depth': 15, 'min_samples_split': 2, 'n_e...	0.628000	0.613386	0.692444	0.650522
12	RandomForest_Importance	XGBoost	{'learning_rate': 0.2, 'max_depth': 7, 'n_esti...	0.627556	0.623388	0.644444	0.633741
13	Mutual_Info	RandomForest	{'max_depth': 15, 'min_samples_split': 2, 'n_e...	0.625333	0.628885	0.611556	0.620099
14	SelectKBest	LogisticRegression	{'C': 1, 'solver': 'liblinear'}	0.622667	0.625683	0.610667	0.618084

	Feature_Selection	Model	Best_Params	Accuracy	Precision	Recall	F1_Score
15	L1_Lasso	DecisionTreeClassifier	{'criterion': 'entropy', 'max_depth': 5, 'min_...	0.592889	0.616760	0.490667	0.546535
16	Mutual_Info	LogisticRegression	{'C': 10, 'solver': 'liblinear'}	0.586222	0.586607	0.584000	0.585301
17	RandomForest_Importance	DecisionTreeClassifier	{'criterion': 'entropy', 'max_depth': None, 'm...	0.581778	0.576285	0.617778	0.596311
18	Mutual_Info	DecisionTreeClassifier	{'criterion': 'entropy', 'max_depth': 10, 'min...	0.576444	0.576649	0.575111	0.575879
19	RFE	CatBoost	{'depth': 3, 'iterations': 100, 'learning_rate...	0.573778	0.564441	0.646222	0.602569
20	RFE	XGBoost	{'learning_rate': 0.1, 'max_depth': 3, 'n_esti...	0.573778	0.564441	0.646222	0.602569
21	RFE	DecisionTreeClassifier	{'criterion': 'gini', 'max_depth': 10, 'min_sa...	0.573778	0.564441	0.646222	0.602569
22	RFE	LogisticRegression	{'C': 0.01, 'solver': 'liblinear'}	0.573778	0.564441	0.646222	0.602569
23	RFE	RandomForest	{'max_depth': 5, 'min_samples_split': 2, 'n_es...	0.568444	0.593674	0.433778	0.501284
24	RandomForest_Importance	LogisticRegression	{'C': 0.01, 'solver': 'liblinear'}	0.513333	0.512755	0.536000	0.524120