Heart Disease Detection Model Part:

Dataset Selected:

https://www.kaggle.com/datasets/iamsouravbanerjee/heart-attack-prediction-dataset

Exploration of Dataset:

This dataset contains health, lifestyle, and demographic information for over 8,700 patients, with the goal of predicting heart attack risk. It includes medical factors such as age, cholesterol, blood pressure, heart rate, diabetes, BMI, triglycerides, and previous heart problems, which are direct indicators of cardiovascular health. It also covers lifestyle habits like smoking, alcohol use, exercise, diet, stress level, sedentary hours, and sleep, which strongly influence the chances of developing heart disease. Demographic details such as sex, income, and family history are also included, as they can provide context about genetic or environmental risk factors.

Better Features Ideas to Add:

HDL and LDL Cholesterol

HDL (High-Density Lipoprotein) is often called the "good cholesterol" because it helps remove harmful cholesterol from the blood.

LDL (Low-Density Lipoprotein) is the "bad cholesterol" that builds up in arteries and causes blockages.

Measuring them separately is better than just total cholesterol because a person could have high total cholesterol but still a healthy balance between HDL and LDL.

Blood Sugar Levels (Fasting Glucose, HbA1c)

Fasting Glucose measures blood sugar after not eating for 8+ hours.

HbA1c shows average blood sugar levels over the last 2–3 months.

These help detect diabetes more accurately, and diabetes is one of the biggest risk factors for heart disease.

Waist-to-Hip Ratio

Instead of just BMI, this looks at how fat is stored in the body.

Fat around the waist ("abdominal fat") is more dangerous for the heart than fat around the hips.

A high waist-to-hip ratio means more risk of clogged arteries and heart attack.

ECG / EKG Test Results

An Electrocardiogram (ECG/EKG) records the electrical activity of the heart.

It can show irregular heartbeats, poor blood flow, or signs of previous heart damage.

This is a direct measurement of heart function, so including it would improve predictions greatly.

Stress Test Results (Treadmill or Exercise Test)

This test measures how the heart performs under physical activity.

It helps detect hidden heart problems that don't appear when a person is resting.

Adding this would show how well the heart can handle stress, which is key to heart attack risk.

Detailed Medication Information

Right now, dataset only has "Medication Use = Yes/No."

But knowing which type (e.g., cholesterol-lowering drugs, blood pressure meds, diabetes meds) can give direct clues about the patient's condition.

For example, someone on blood pressure medication may still have controlled BP, which lowers risk.

Based on dataset we selected two models :

As dataset has medical features (cholesterol, blood pressure, heart rate, diabetes, BMI, triglycerides), lifestyle features (smoking, alcohol, diet, exercise, sedentary hours, stress, sleep), and demographic information (sex, income, family history, country). The target variable is Heart Attack Risk (binary classification).

This is a tabular dataset with a mix of numerical (age, cholesterol, BMI, etc.) and categorical variables (sex, diet). Such data is typically best handled by tree-based models (like Gradient Boosting), but neural networks can also be applied.

1st Model Preferred: **Gradient Boosting (XGBoost, LightGBM, CatBoost)**

How it works

- Gradient Boosting builds a series of decision trees, where each new tree corrects the mistakes made by the previous ones.
- Instead of training many independent trees like Random Forest, it trains trees sequentially to minimize error.
- It optimizes using a gradient descent approach (hence the name).

Why it fits for this dataset

- Handles both continuous values (e.g., cholesterol, triglycerides) and categorical variables (diet, sex).
- Works well on imbalanced medical data, since boosting focuses on harder-to-predict cases.
- Provides feature importance scores, so you can explain which factors (BMI, cholesterol, exercise, etc.) drive predictions.
- Often the best performing algorithm for structured/tabular health datasets.

Pros

- High accuracy, especially on tabular health records.
- Can handle missing data (some versions like CatBoost do this naturally).
- Explains feature importance → useful in medical research.

Cons

- Training can be slower than Random Forest.
- Needs hyperparameter tuning (learning rate, depth, estimators) for best results.

2nd Model Preferred: **Neural Networks (Multi-Layer Perceptron for Tabular Data)**

How it works

- A neural network consists of layers of nodes (neurons) that apply transformations to input features.
- For tabular data, you'd feed in all 25 features (after preprocessing), and hidden layers would learn complex patterns between them.
- Output layer would be a single neuron with sigmoid activation for binary classification (risk/no risk).

Why it fits for this dataset

- Your dataset is large enough (~8,700 rows) to train a small neural network.
- Can learn non-linear relationships that logistic regression or linear models can't.
- If you add more complex health signals (like ECG time-series or imaging data), neural networks outperform tree-based models.

Pros

- Flexible, can learn complex feature interactions.
- Works well if you expand your dataset (ECG, medical images, continuous monitoring data).

- Can be combined with embeddings (for categorical variables like diet or sex).

Cons

- Needs careful preprocessing:
  - Normalize numerical features (cholesterol, triglycerides, etc.).
  - One-hot encode categorical features.
- Less interpretable than Gradient Boosting (a black box).
- With only tabular data, may not outperform boosting models unless the dataset grows larger or becomes multi-modal (text, signals, images).

EDA(EXPLORATORY DATA ANALYSIS) OF DATASET:

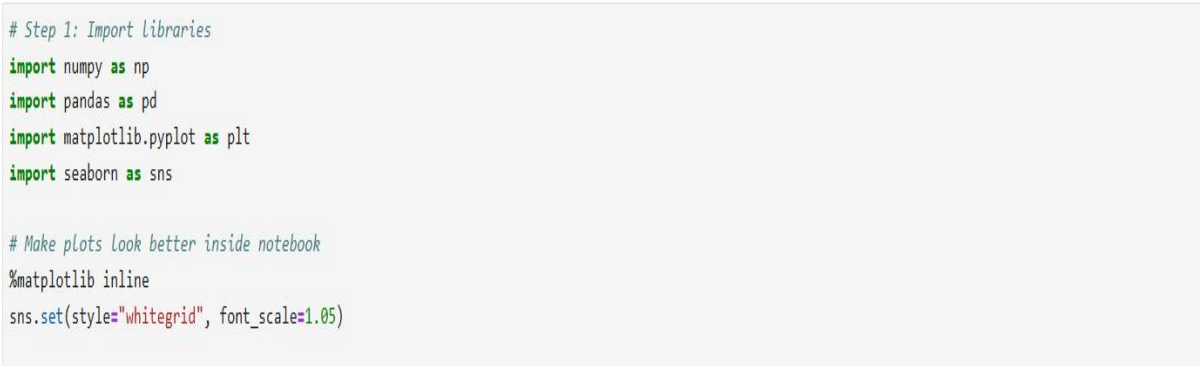
Step 1: Import Libraries

We started by importing essential Python libraries like Pandas, NumPy, Matplotlib, and Seaborn. These tools provide the backbone for data cleaning, statistical analysis, and visualization. This step ensures we have everything ready to explore the dataset effectively.

```
[2]: # Step 1: Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Make plots look better inside notebook
%matplotlib inline
sns.set(style="whitegrid", font_scale=1.05)
```

Step 2: Load Dataset

We loaded the CSV file into a Pandas DataFrame. This step allows us to organize the raw dataset into a structured table that we can manipulate easily. Having the data inside a DataFrame is the first step to performing any analysis or preprocessing.



```
# Step 1: Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Make plots look better inside notebook
%matplotlib inline
sns.set(style="whitegrid", font_scale=1.05)
```

Rows, Columns: (8763, 26)

[4]:

	Patient ID	Age	Sex	Cholesterol	Blood Pressure	Heart Rate	Diabetes	Family History	Smoking	Obesity	...	Sedentary Hours Per Day	Income	BMI	Triglycerides	Physical Activity Days Per Week	Sleep Hours Per Day	Cour
0	BMW7812	67	Male	208	158/88	72	0	0	1	0	...	6.615001	261404	31.251233	286	0	6	Argen
1	CZE1114	21	Male	389	165/93	98	1	1	1	1	...	4.963459	285768	27.194973	235	1	7	Can
2	BNI9906	21	Female	324	174/99	72	1	0	0	0	...	9.463426	235282	28.176571	587	4	4	Fra
3	JLN3497	84	Male	383	163/100	73	1	1	1	0	...	7.648981	125640	36.464704	378	3	4	Can
4	GFO8847	66	Male	318	91/88	93	1	1	1	1	...	1.514821	160555	21.809144	231	1	5	Thail

5 rows × 26 columns

Step 3: Dataset Overview

Using `.head()` and `.info()`, we previewed the dataset and checked data types. This helps us confirm the dataset loaded correctly and gives us a first impression of the features. It also identifies whether the columns are numerical or categorical, which affects modeling decisions.

```
# Step 3: Basic info and summary
print("=== Dataset Info ===")
df.info()

print("\n=== Summary statistics for numerical columns ===")
df.describe().T
```

```
=== Dataset 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)
```

```
=== Summary statistics for numerical columns ===
```

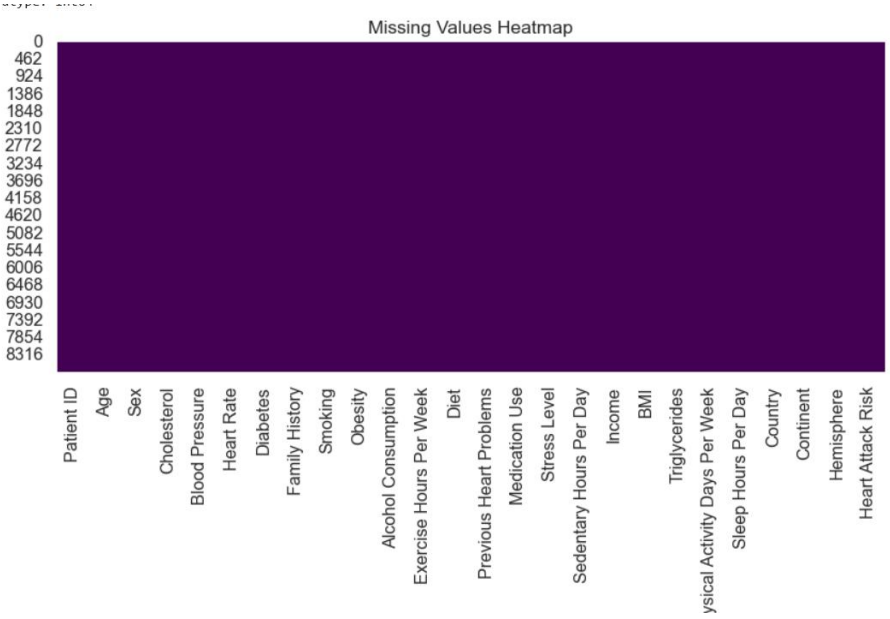
	count	mean	std	min	25%	50%	75%	max
Age	8763.0	53.707977	21.249509	18.000000	35.000000	54.000000	72.000000	90.000000
Cholesterol	8763.0	259.877211	80.863276	120.000000	192.000000	259.000000	330.000000	400.000000
Heart Rate	8763.0	75.021682	20.550948	40.000000	57.000000	75.000000	93.000000	110.000000
Diabetes	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 Per Week	8763.0	3.489672	2.282687	0.000000	2.000000	3.000000	5.000000	7.000000
Sleep Hours Per Day	8763.0	7.023508	1.988473	4.000000	5.000000	7.000000	9.000000	10.000000

Step 4: Missing & Infinite Values

We checked for missing values and replaced any infinite values with `NaN`. This ensures the dataset is clean and avoids errors in statistical calculations. Data quality directly affects model performance, so this step ensures reliability in training.

```python
# Step 4: Check missing values
missing = df.isnull().sum()
print("Missing values per column:\n", missing)

# Heatmap to visualize missing values
plt.figure(figsize=(10,4))
sns.heatmap(df.isnull(), cbar=False, cmap="viridis")
plt.title("Missing Values Heatmap")
plt.show()
```

```
Missing values per column:
 Patient ID                        0
Age                                0
Sex                                0
Cholesterol                        0
Blood Pressure                     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
Continent                          0
Hemisphere                         0
Heart Attack Risk                  0
dtype: int64
```

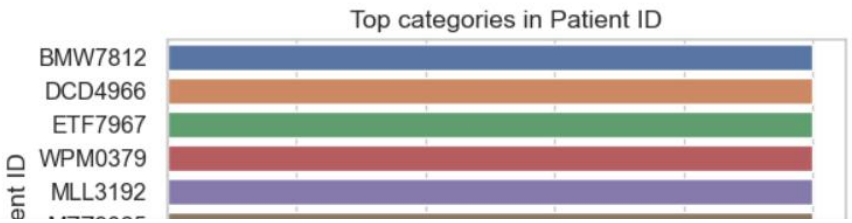


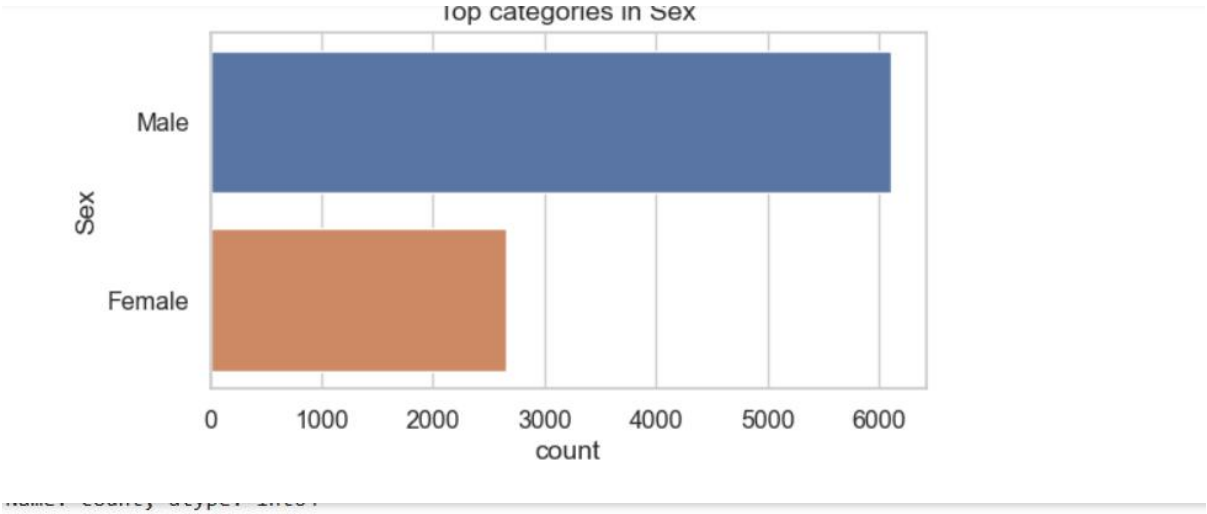
Step 5: Explore Categorical Columns

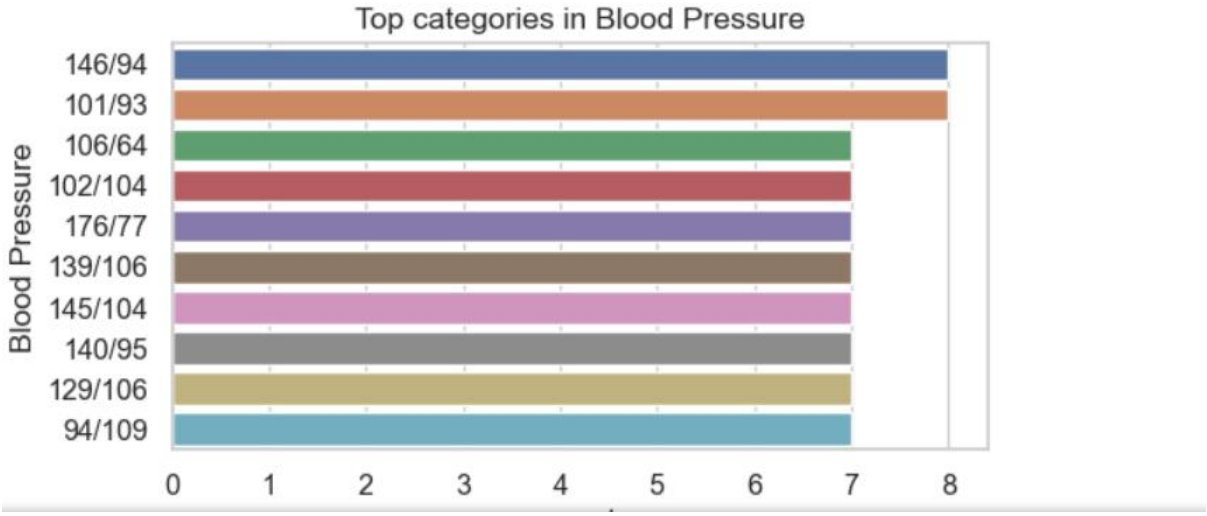
We selected categorical columns and plotted count charts using Seaborn. This step shows how values are distributed across categories like gender, chest pain type, or exercise-induced angina. Understanding category frequency is important to check class imbalance and decide feature importance.

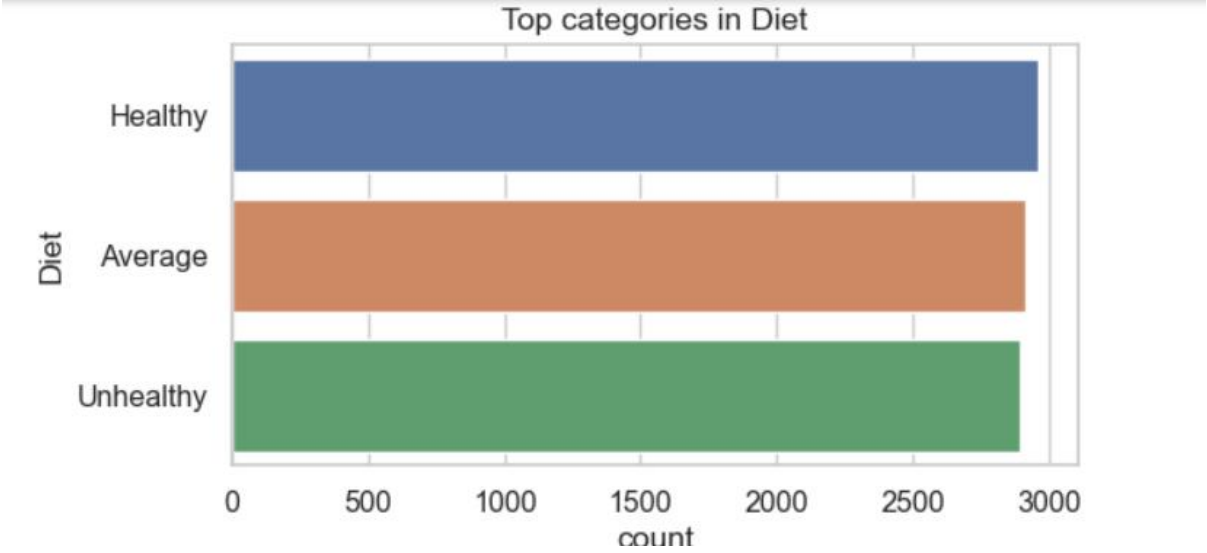
```
# Step 5: Explore categorical columns
cat_cols = df.select_dtypes(include=['object']).columns
for col in cat_cols:
    print(f"\nColumn: {col}")
    print(df[col].value_counts().head())

    plt.figure(figsize=(6,3))
    sns.countplot(y=col, data=df, order=df[col].value_counts().index[:10])
    plt.title(f"Top categories in {col}")
    plt.show()
```

```
Column: Patient ID
Patient ID
BMW7812    1
DCD4966    1
ETF7967    1
WPM0379    1
MLL3192    1
Name: count, dtype: int64
```

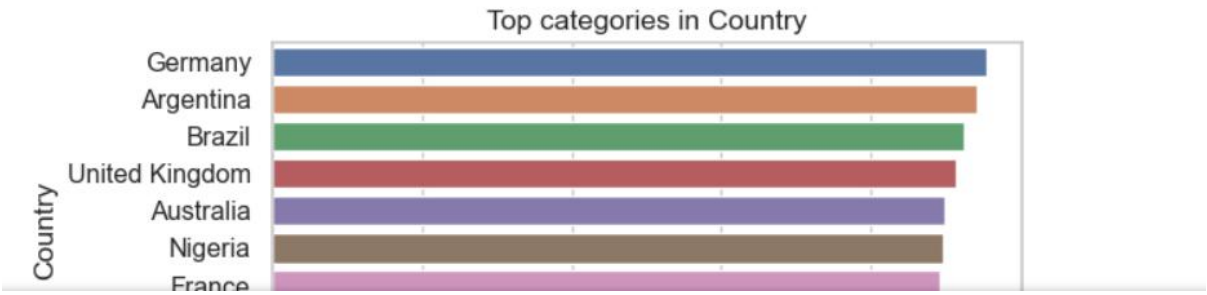



Top categories in Sex
Male
Female
Sex
0
1000
2000
3000
4000
5000
6000
count

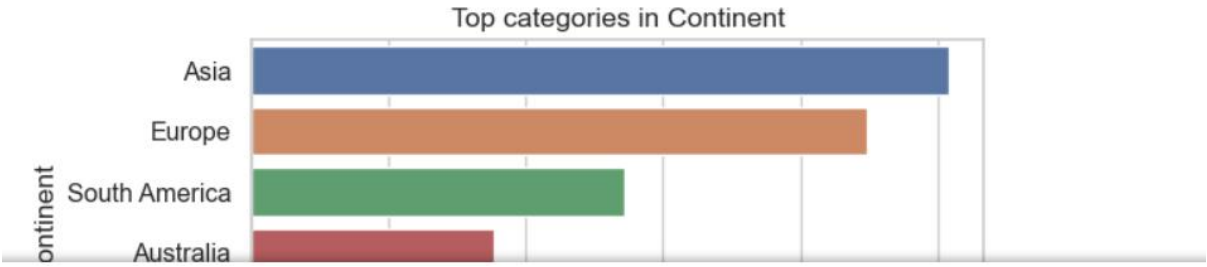

Top categories in Blood Pressure
146/94
101/93
106/64
102/104
176/77
139/106
145/104
140/95
129/106
94/109
Blood Pressure
0
1
2
3
4
5
6
7
8


Top categories in Diet
Healthy
Average
Unhealthy
Diet
0
500
1000
1500
2000
2500
3000
count

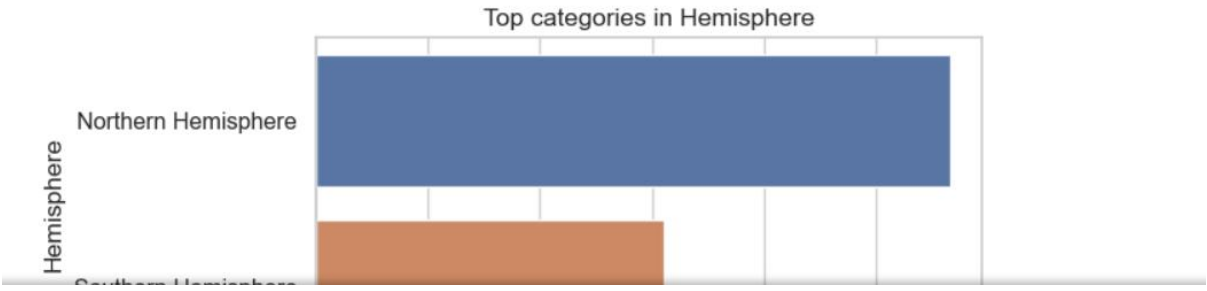
```
Country
Germany           477
Argentina         471
Brazil            462
United Kingdom    457
Australia         449
Name: count, dtype: int64
```




```
Column: Continent
Continent
Asia             2543
Europe           2241
South America    1362
Australia         884
Africa            873
Name: count, dtype: int64
```



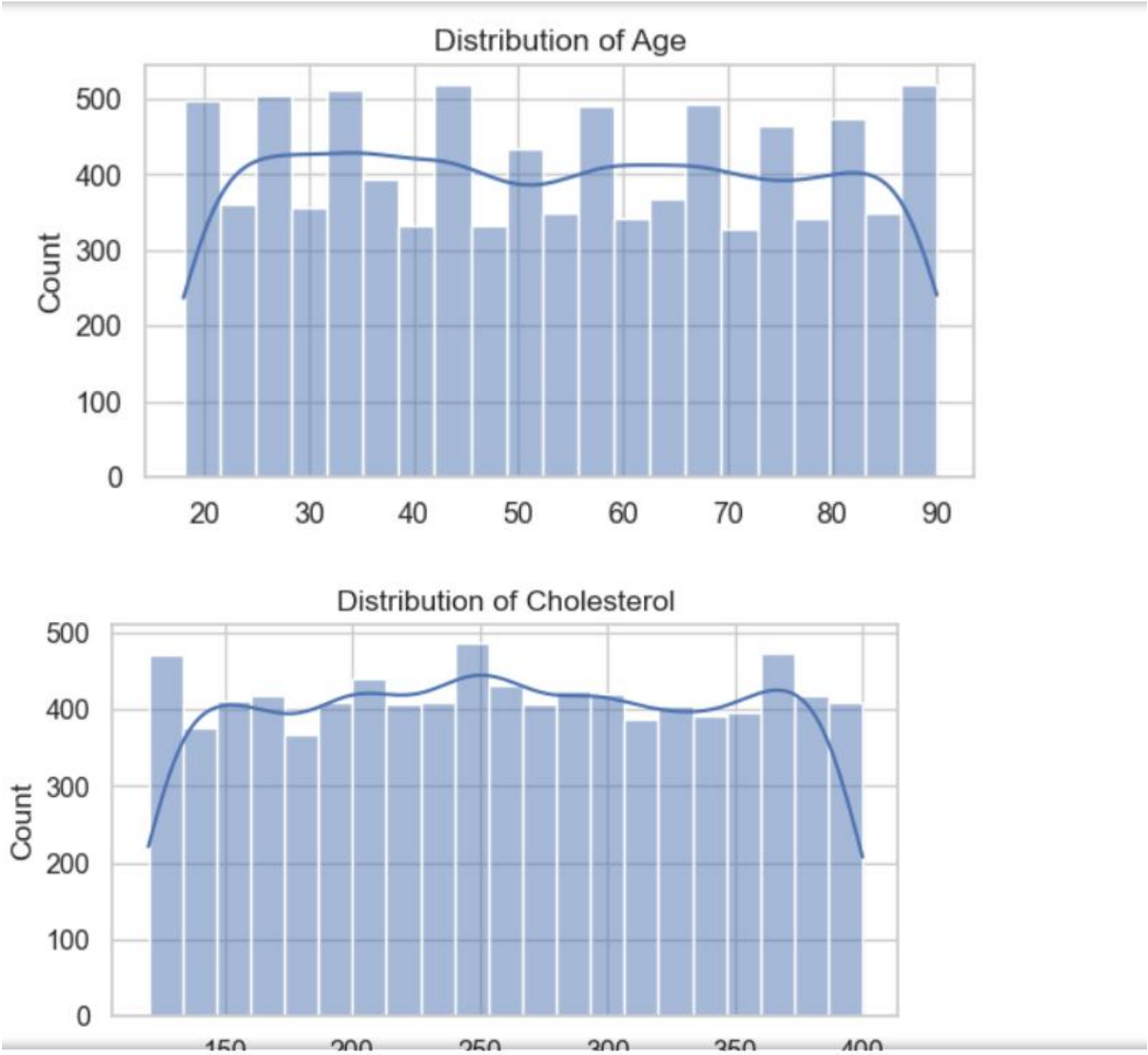

```
Column: Hemisphere
Hemisphere
Northern Hemisphere    5660
Southern Hemisphere    3103
Name: count, dtype: int64
```

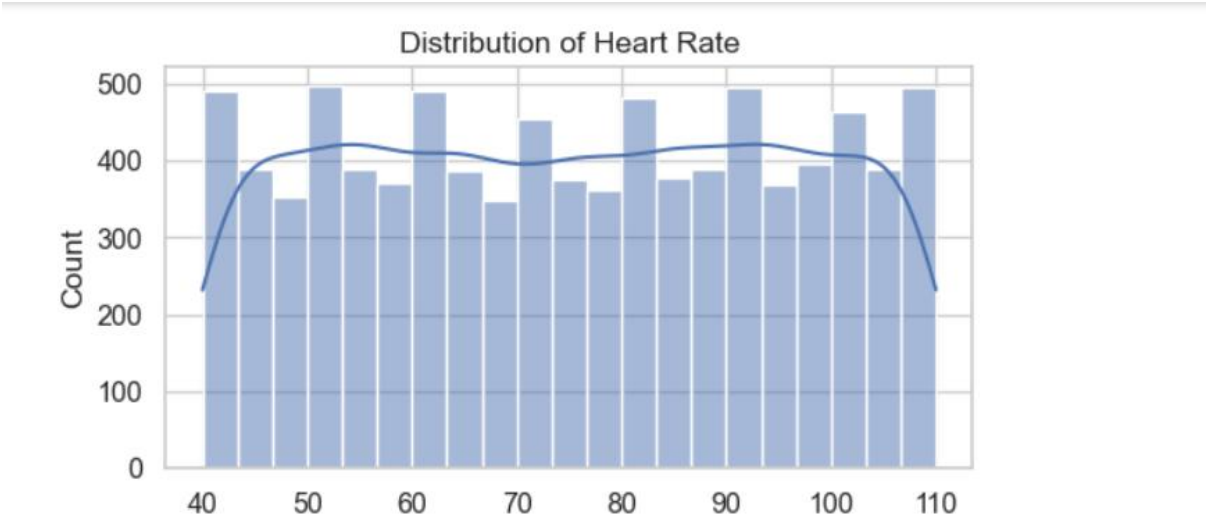


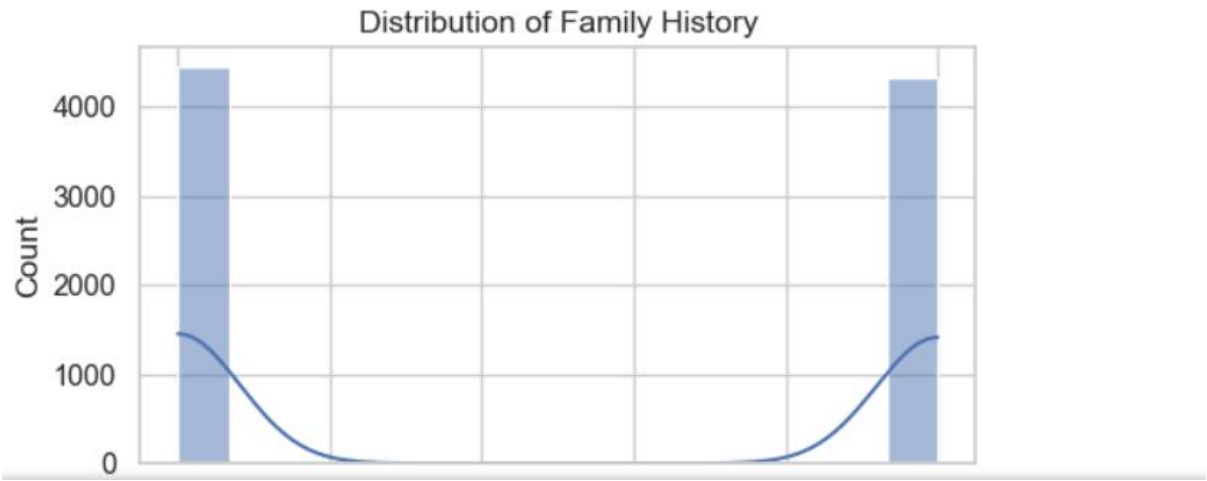
Step 6: Explore Numerical Columns

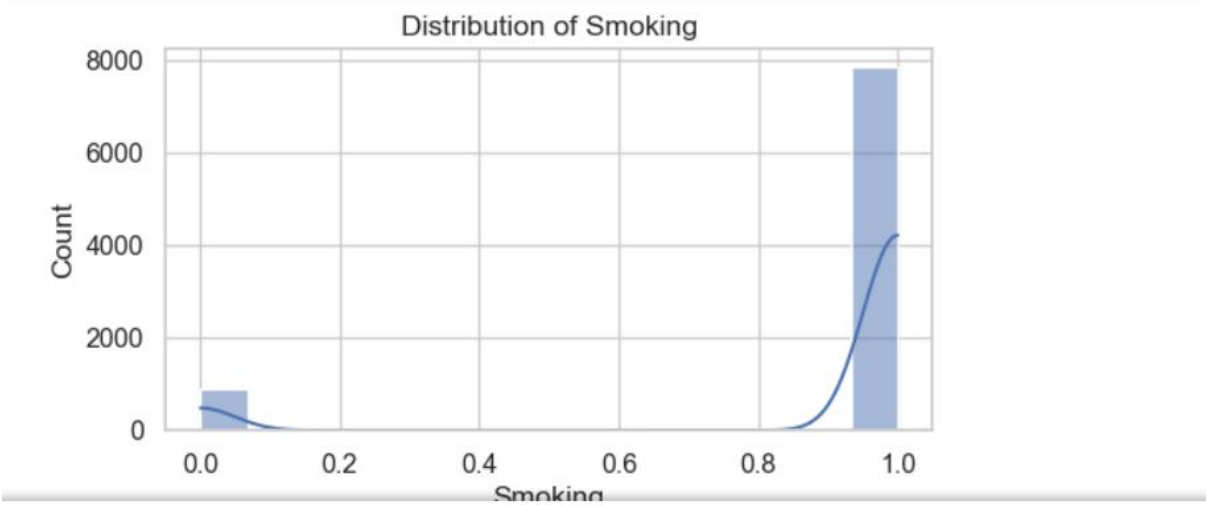
In this step, we selected numeric columns and plotted their distributions using histograms with KDE curves. These plots show how values like age, cholesterol, and blood pressure are spread across patients. Understanding the distribution helps identify normal vs. skewed variables and points out which features may require scaling or transformation before training.

```
# Step 6: Explore numerical columns
num_cols = df.select_dtypes(include=[np.number]).columns
for col in num_cols[:6]:   # show first 6 numeric columns
    plt.figure(figsize=(6,3))
    sns.histplot(df[col], kde=True)
    plt.title(f"Distribution of {col}")
    plt.show()
```





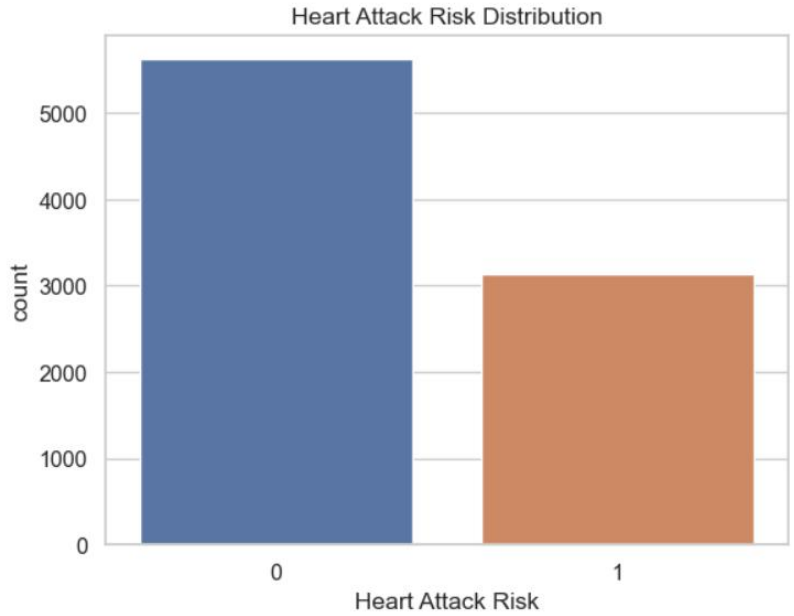




Step 7: Target Variable Distribution (Heart Attack Risk)

Here we visualized the target column, **Heart Attack Risk**, using a count plot. This shows how many patients are classified as at risk versus not at risk. Checking this distribution is

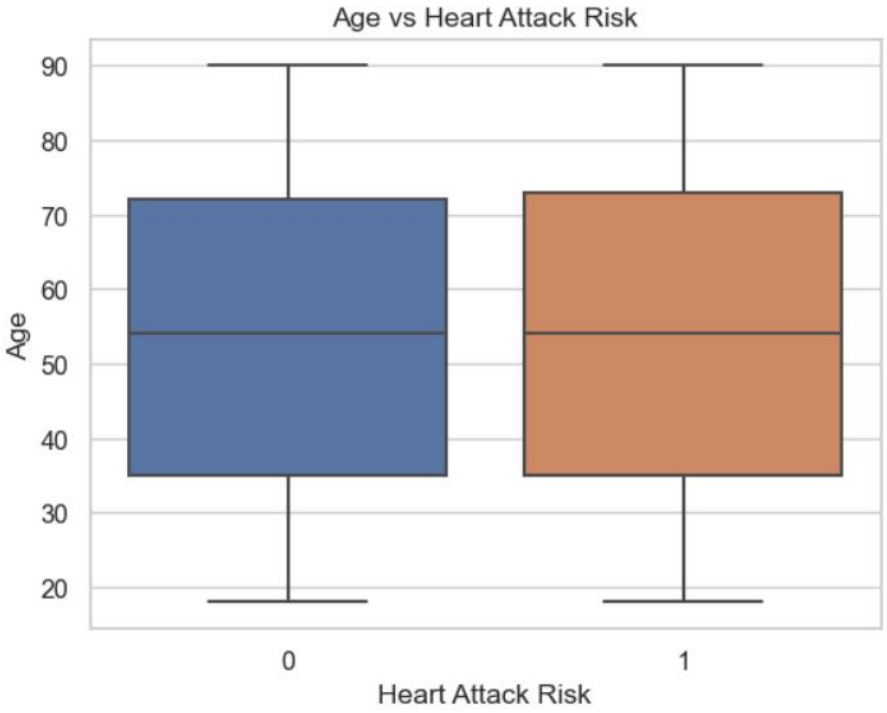
crucial because if the dataset is imbalanced (e.g., far more “no risk” than “risk” cases), the model may become biased toward the majority class. If imbalance exists, resampling or class weighting will be needed to build a fair predictive model.

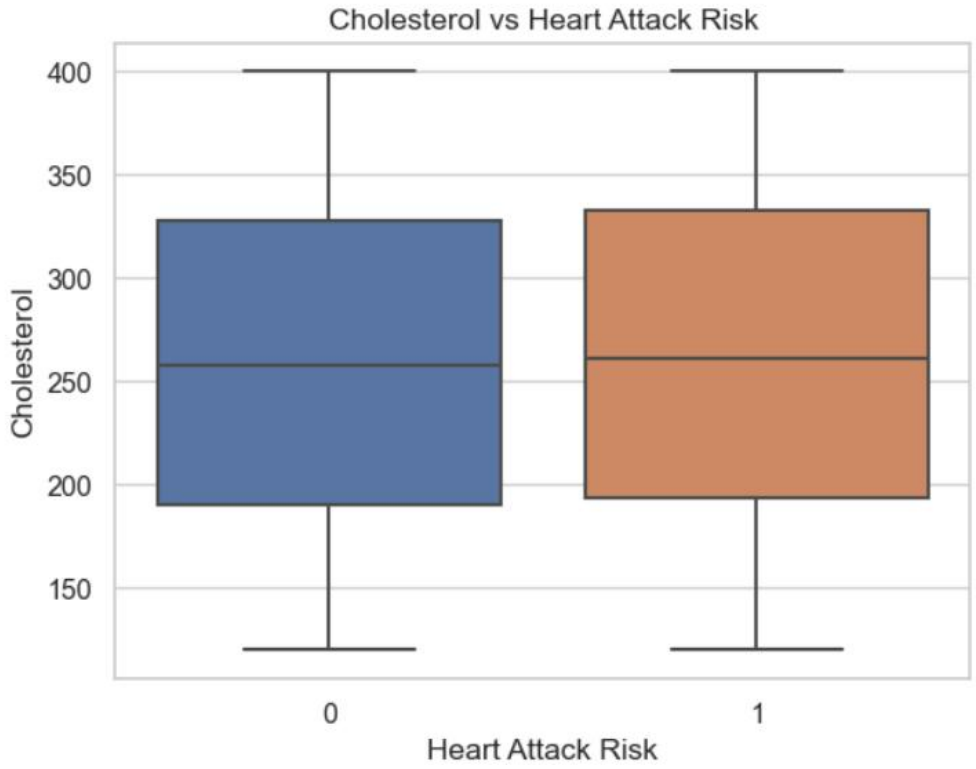


Step 8: Relationship Between Numeric Features and Target

We used boxplots to compare how numeric features (like **Age** and **Cholesterol**) differ between patients with and without heart attack risk. These plots reveal whether higher or lower values in these features are associated with greater risk. For example, if patients with higher cholesterol show a higher frequency of heart attack risk, that makes cholesterol a strong predictive feature.

```
# Step 8: Relationship between numeric features and target
sns.boxplot(x="Heart Attack Risk", y="Age", data=df)
plt.title("Age vs Heart Attack Risk")
plt.show()

sns.boxplot(x="Heart Attack Risk", y="Cholesterol", data=df)
plt.title("Cholesterol vs Heart Attack Risk")
plt.show()
```

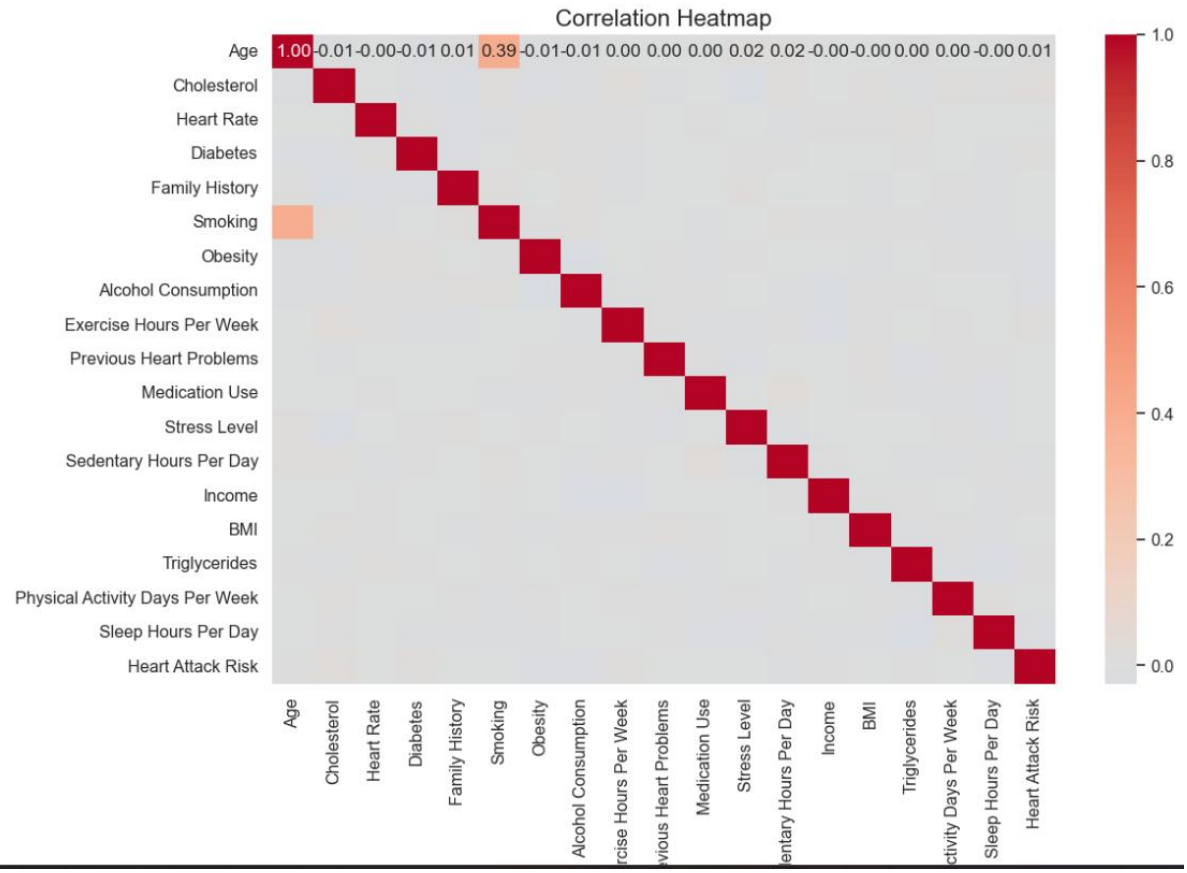




Step 9: Correlation Heatmap

We plotted a heatmap to visualize the correlation between all numeric features. This helps identify which variables move together and which features are most strongly related to the target variable (Heart Attack Risk). Strong positive or negative correlations highlight

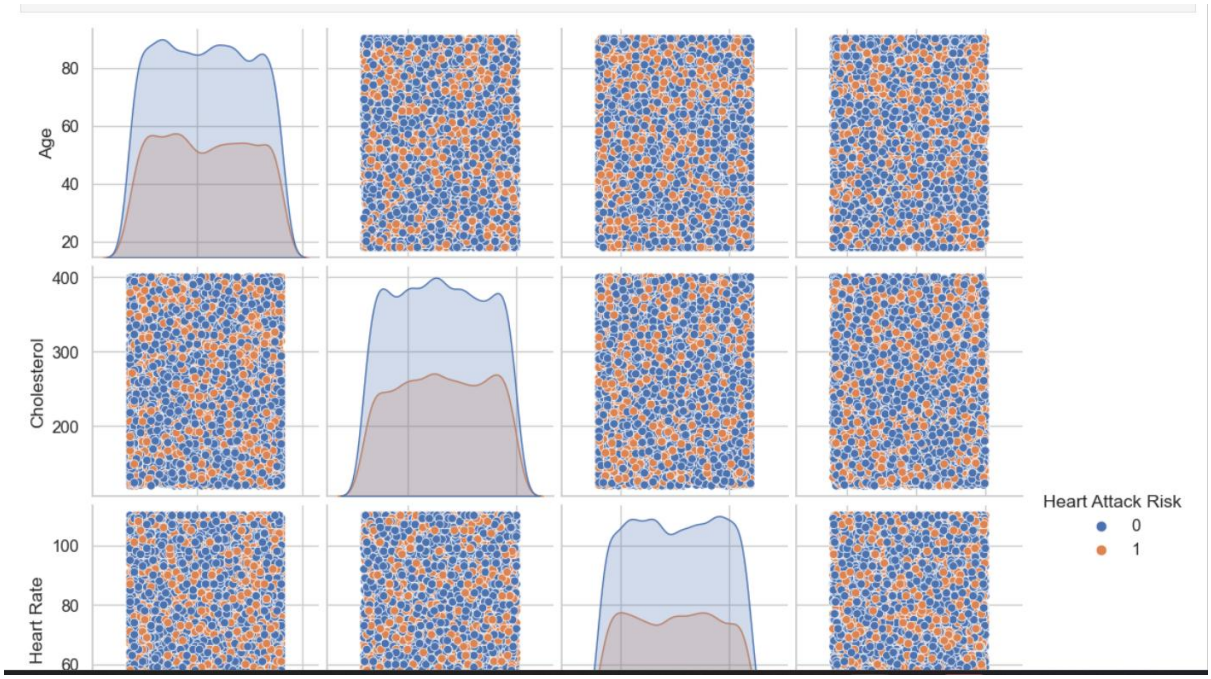
potential predictors, while very high correlations between independent features may indicate redundancy (multicollinearity), which should be handled before model training.

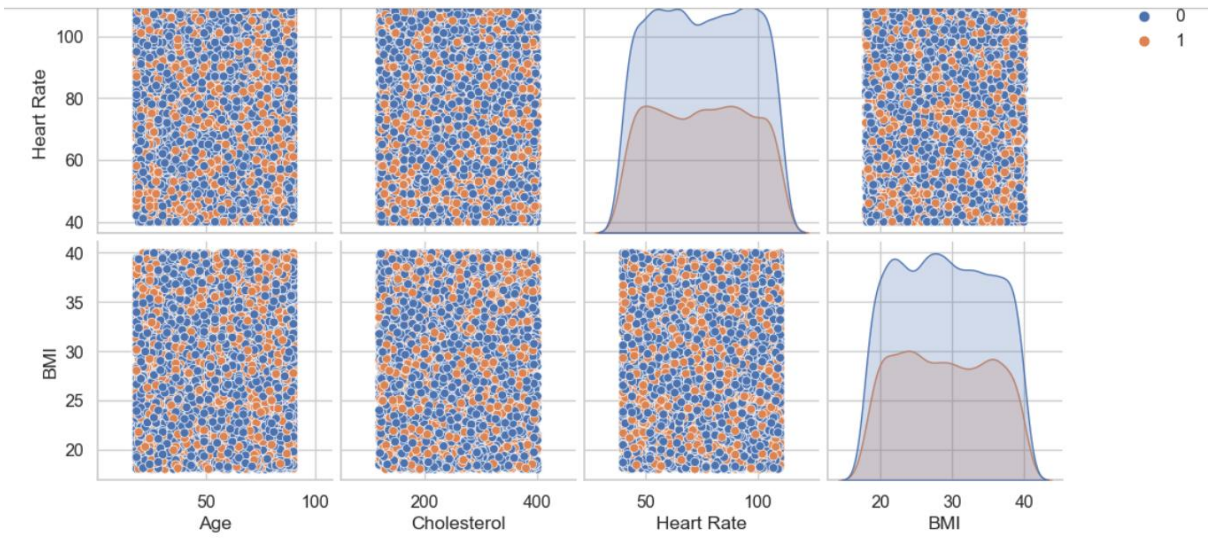


Step 10: Pairplot for Key Features

We created a pairplot of selected features (**Age, Cholesterol, Heart Rate, and BMI**) against the target variable (**Heart Attack Risk**). This visualization helps us see how these features interact with each other and whether clusters form between patients at risk and those not at risk. By examining these scatter and density plots, we gain a deeper understanding of which features are most discriminative and useful for building predictive models.

```
# Step 10: Pairplot for key features
sns.pairplot(df[['Age','Cholesterol','Heart Rate','BMI','Heart Attack Risk']],
             hue="Heart Attack Risk", diag_kind="kde")
plt.show()
```


Age
Cholesterol
Heart Rate
Heart Attack Risk
0
1


Heart Rate
BMI
Age
Cholesterol
Heart Rate
BMI
0
1