In [29]:

```
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.impute import SimpleImputer
import joblib
```

In [30]:

```
import re
df = pd.read_csv("cardio_train.csv", sep=None, engine="python")
df.columns = (df.columns.str.strip()
                        .str.lower()
                        .str.replace(r"[^a-z0-9]+", "_", regex=True)
                        .str.strip("_"))
rename_map = {
    "sex":"gender","systolic":"ap_hi","systolic_bp":"ap_hi","aphi":"ap_hi",
    "diastolic":"ap_lo","diastolic_bp":"ap_lo","aplo":"ap_lo",
    "alcohol":"alco","glucose":"gluc","cardiovascular_disease":"cardio",
    "height_cm":"height","weight_kg":"weight"
}
rename_map = {k:v for k,v in rename_map.items() if k in df.columns and v not in df.columns}
df = df.rename(columns=rename_map)
print("Columns:", df.columns.tolist())
df.head()
```

```
Columns: ['id', 'age', 'gender', 'height', 'weight', 'ap_hi', 'ap_lo', 'cholesterol', 'gluc', 'smoke', 'alco', 'active', 'cardio']
```

Out[30]:

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	0	18393	2	168	62.0	110	80	1	1	0	0	1	0
1	1	20228	1	156	85.0	140	90	3	1	0	0	1	1
2	2	18857	1	165	64.0	130	70	3	1	0	0	0	1
3	3	17623	2	169	82.0	150	100	1	1	0	0	1	1
4	4	17474	1	156	56.0	100	60	1	1	0	0	0	0

In [60]:

```
for c in ["age","gender","height","weight","ap_hi","ap_lo",
          "cholesterol","gluc","smoke","alco","active","cardio"]:
    if c in df.columns:
        df[c] = pd.to_numeric(df[c], errors="coerce")

if "age" in df.columns:
    if df["age"].median() > 150:
        df["age_years"] = (df["age"] / 365.25).round(0)
    else:
        df["age_years"] = df["age"]
else:
    df["age_years"] = (df["id"] % 60) + 20
    df["age"] = df["age_years"]

if {"height","weight"}.issubset(df.columns):
    h_m = df["height"] / 100.0
    df["BMI"] = df["weight"] / (h_m ** 2)
    df.loc[(h_m == 0) | (h_m.isna()), "BMI"] = np.nan
else:
    df["BMI"] = np.nan

if {"ap_hi","ap_lo"}.issubset(df.columns):
    swap_mask = df["ap_lo"] > df["ap_hi"]
    df.loc[swap_mask, ["ap_hi","ap_lo"]] = df.loc[swap_mask, ["ap_lo","ap_hi"]].values
    df["ap_hi"] = df["ap_hi"].clip(60, 250)
    df["ap_lo"] = df["ap_lo"].clip(30, 150)

if "height" in df.columns:
    df["height"] = df["height"].clip(120, 230)
if "weight" in df.columns:
    df["weight"] = df["weight"].clip(30, 200)

num_cols = df.select_dtypes(include=[np.number]).columns
imp = SimpleImputer(strategy="median")
df[num_cols] = imp.fit_transform(df[num_cols])

print("After preprocessing shape:", df.shape)
df[["id","age","age_years","BMI","ap_hi","ap_lo"]].head()
```

```
After preprocessing shape: (70000, 15)
```

Out[60]:

	id	age	age_years	BMI	ap_hi	ap_lo
0	0.0	18393.0	50.0	21.967120	110.0	80.0
1	1.0	20228.0	55.0	34.927679	140.0	90.0
2	2.0	18857.0	52.0	23.507805	130.0	70.0
3	3.0	17623.0	48.0	28.710479	150.0	100.0
4	4.0	17474.0	48.0	23.011177	100.0	60.0

In [32]:

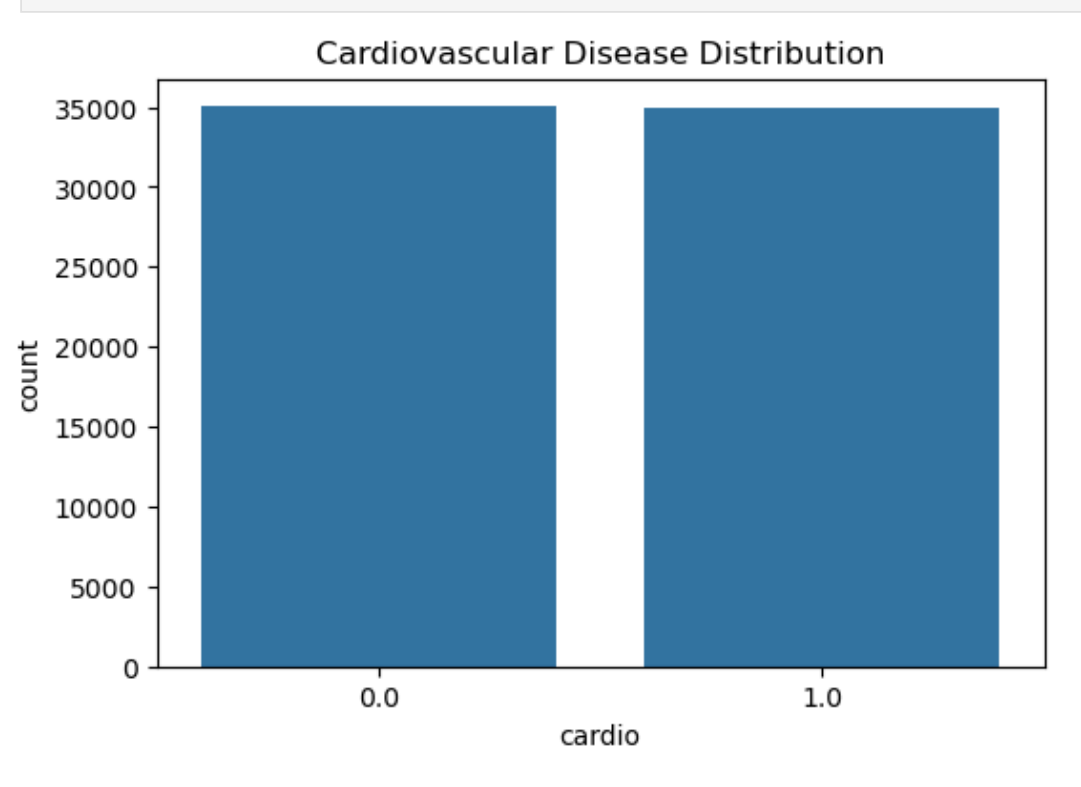
```
df.describe()
```

Out[32]:

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio	age_years	BMI
count	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000
mean	49972.419900	19468.865814	1.349571	164.382000	74.206633	128.311843	82.014171	1.366871	1.226457	0.088129	0.053771	0.803729	0.499700	53.303157	27.556513
std	28851.302323	2467.251667	0.476838	8.018696	14.392385	22.209822	12.414977	0.680250	0.572270	0.283484	0.225568	0.397179	0.500003	6.760171	6.091511
min	0.000000	10798.000000	1.000000	120.000000	30.000000	60.000000	30.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	30.000000	3.471784
25%	25006.750000	17664.000000	1.000000	159.000000	65.000000	120.000000	80.000000	1.000000	1.000000	0.000000	0.000000	1.000000	0.000000	48.000000	23.875115
50%	50001.500000	19703.000000	1.000000	165.000000	72.000000	120.000000	80.000000	1.000000	1.000000	0.000000	0.000000	1.000000	0.000000	54.000000	26.374068
75%	74889.250000	21327.000000	2.000000	170.000000	82.000000	140.000000	90.000000	2.000000	1.000000	0.000000	0.000000	1.000000	1.000000	58.000000	30.222222
max	99999.000000	23713.000000	2.000000	230.000000	200.000000	250.000000	150.000000	3.000000	3.000000	1.000000	1.000000	1.000000	1.000000	65.000000	298.666667

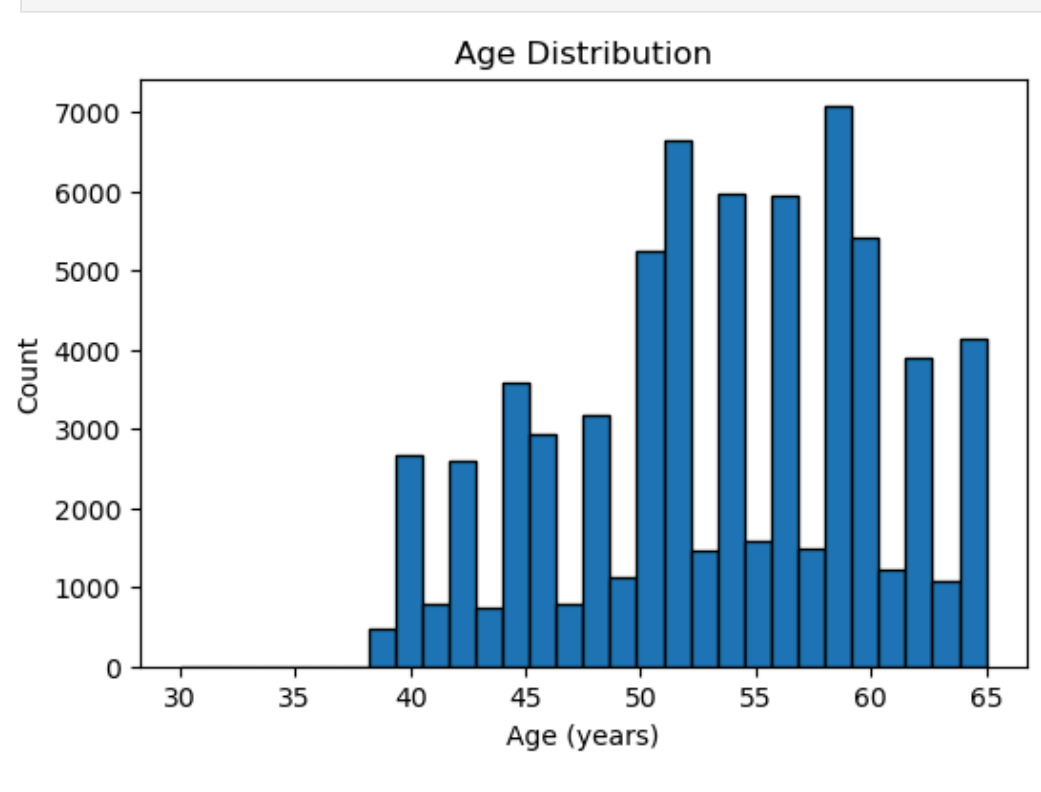
In [33]:

```
plt.figure(figsize=(6,4))
sns.countplot(x="cardio", data=df)
plt.title("Cardiovascular Disease Distribution")
plt.show()
```



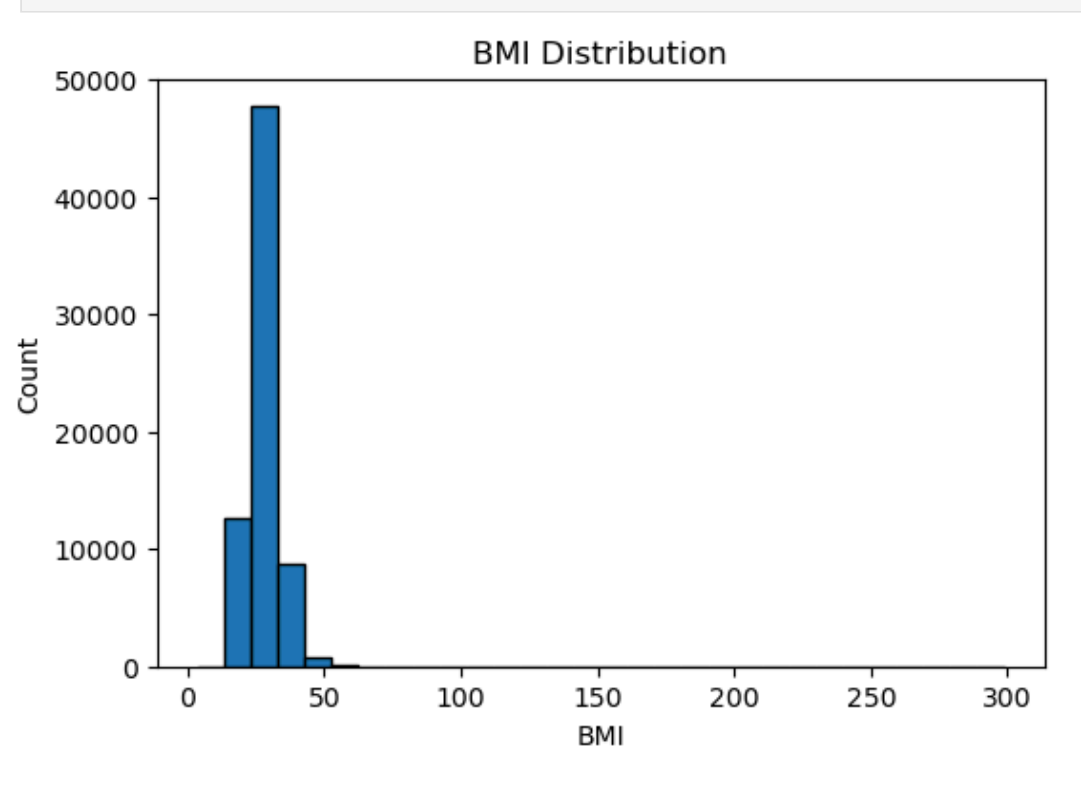
In [34]:

```
plt.figure(figsize=(6,4))
plt.hist(df["age_years"], bins=30, edgecolor="black")
plt.xlabel("Age (years)")
plt.ylabel("Count")
plt.title("Age Distribution")
plt.show()
```



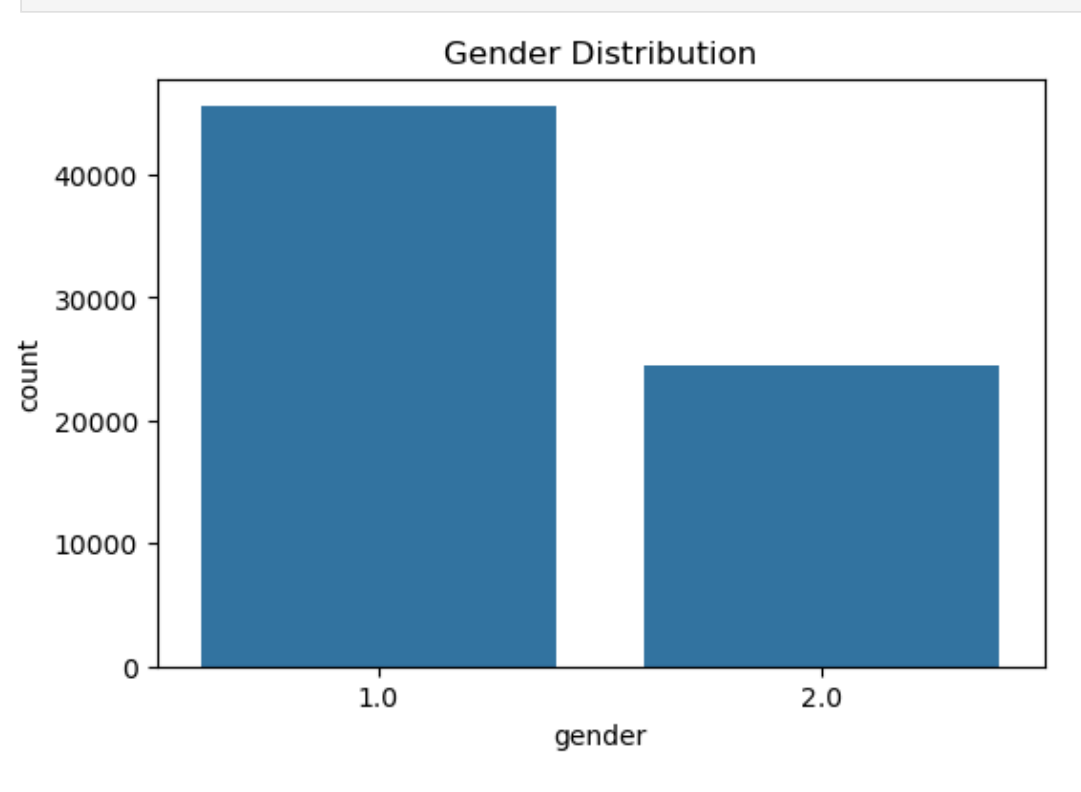
In [35]:

```
plt.figure(figsize=(6,4))
plt.hist(df["BMI"], bins=30, edgecolor="black")
plt.xlabel("BMI")
plt.ylabel("Count")
plt.title("BMI Distribution")
plt.show()
```



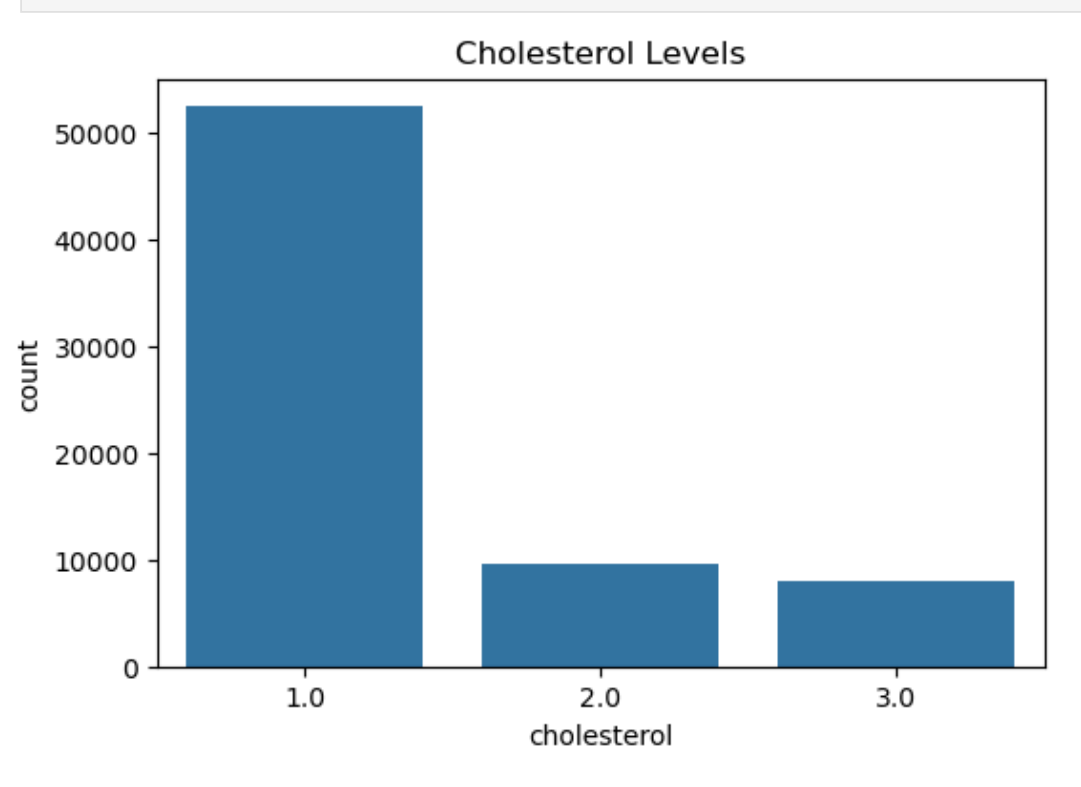
In [8]:

```
plt.figure(figsize=(6,4))
sns.countplot(x="gender", data=df)
plt.title("Gender Distribution")
plt.show()
```



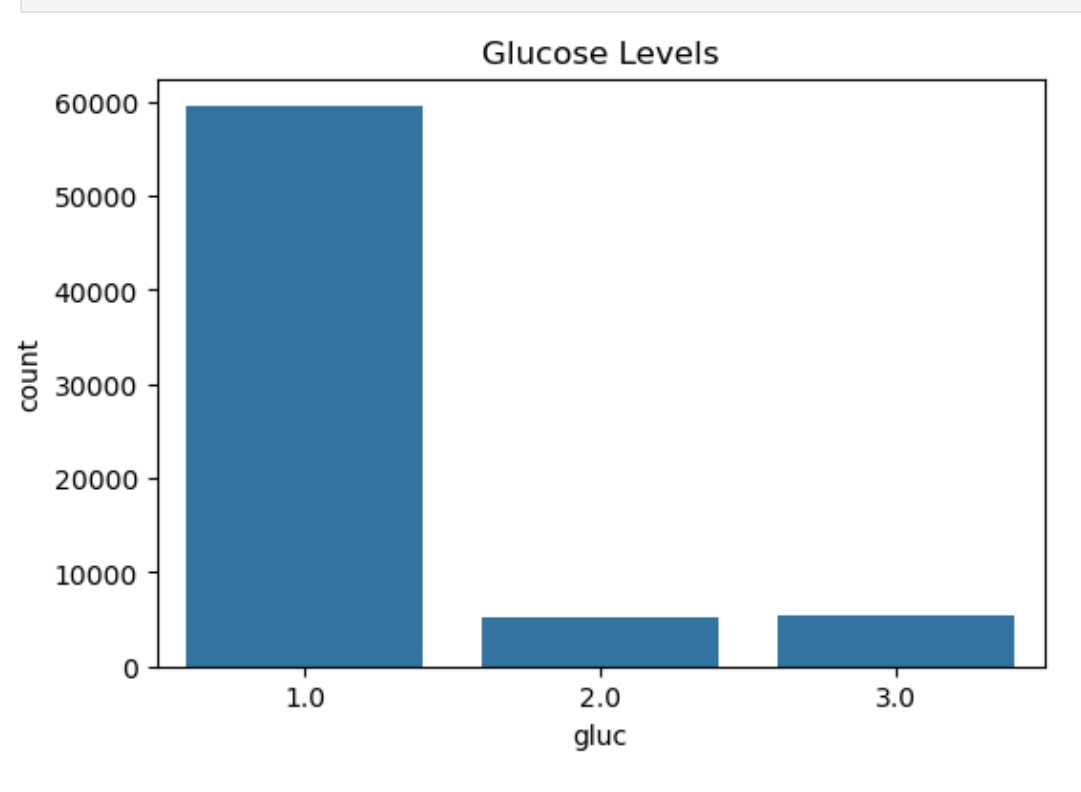
In [36]:

```
plt.figure(figsize=(6,4))
sns.countplot(x="cholesterol", data=df)
plt.title("Cholesterol Levels")
plt.show()
```



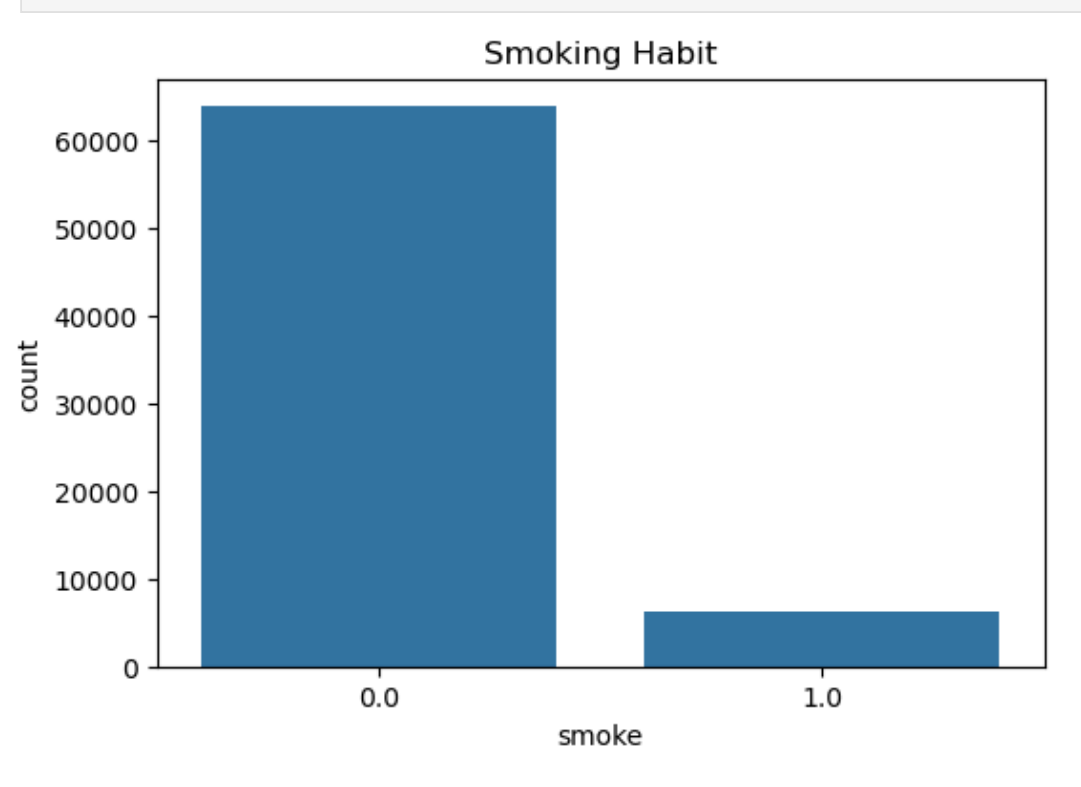
In [10]:

```
plt.figure(figsize=(6,4))
sns.countplot(x="gluc", data=df)
plt.title("Glucose Levels")
plt.show()
```



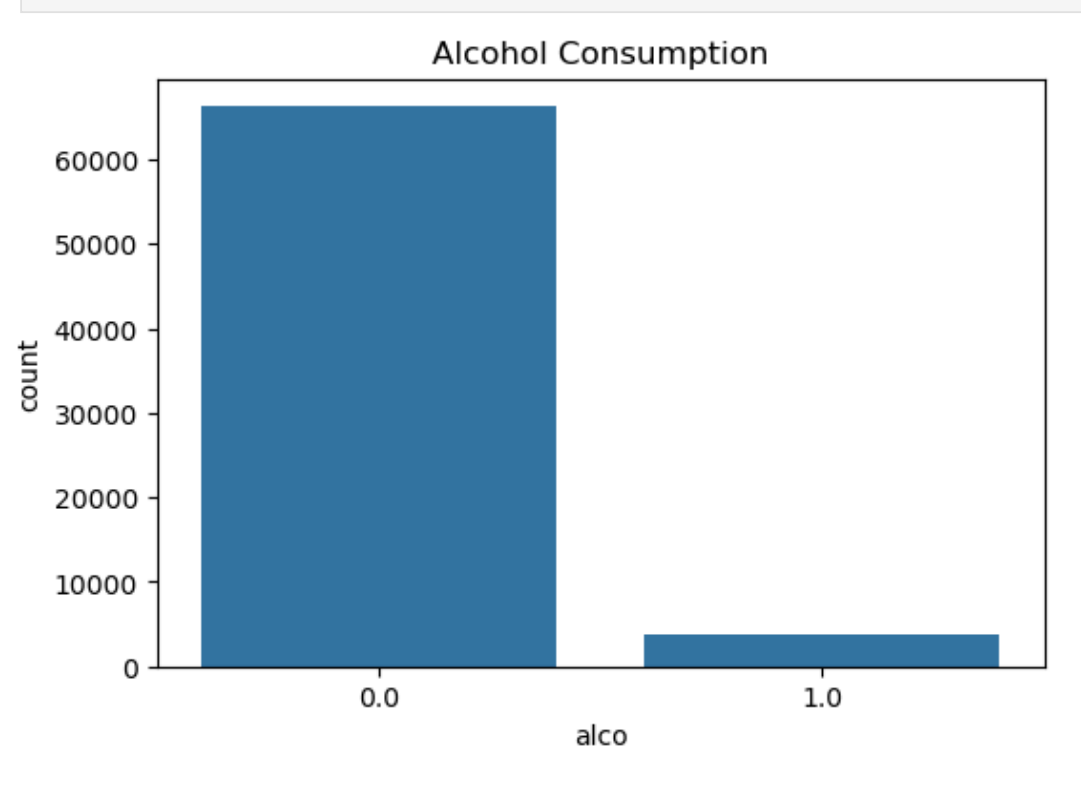
In [37]:

```
plt.figure(figsize=(6,4))
sns.countplot(x="smoke", data=df)
plt.title("Smoking Habit")
plt.show()
```



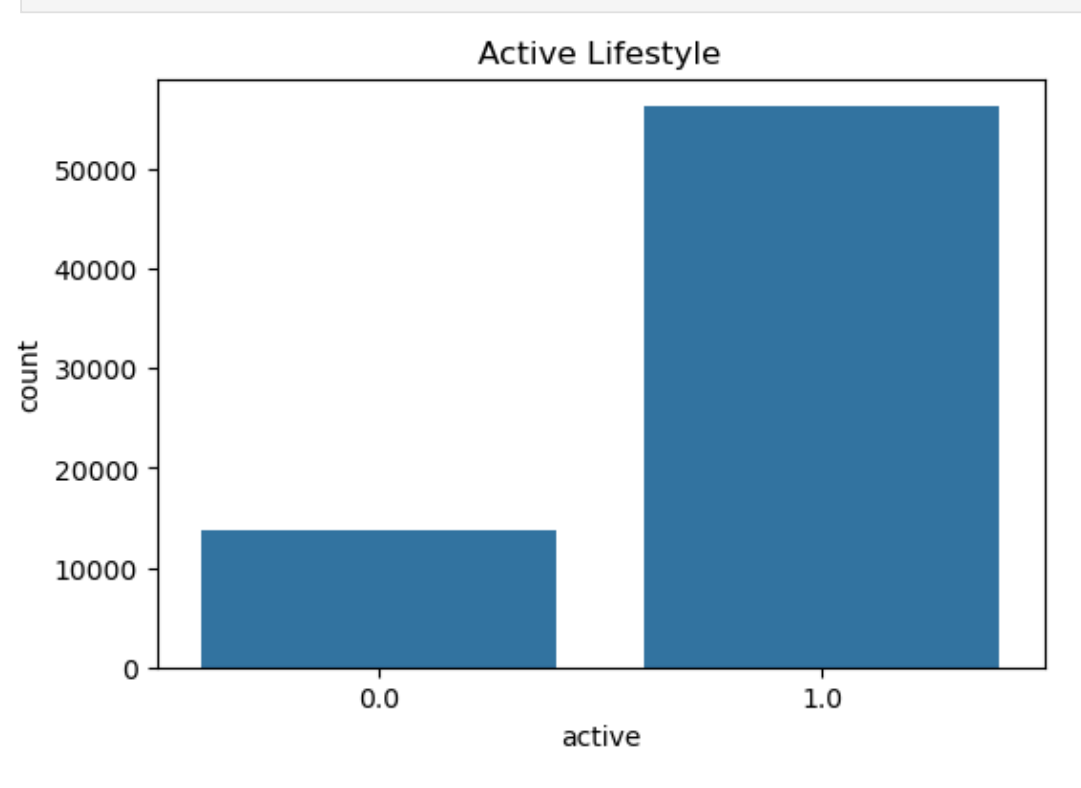
In [12]:

```
plt.figure(figsize=(6,4))
sns.countplot(x="alco", data=df)
plt.title("Alcohol Consumption")
plt.show()
```



In [38]:

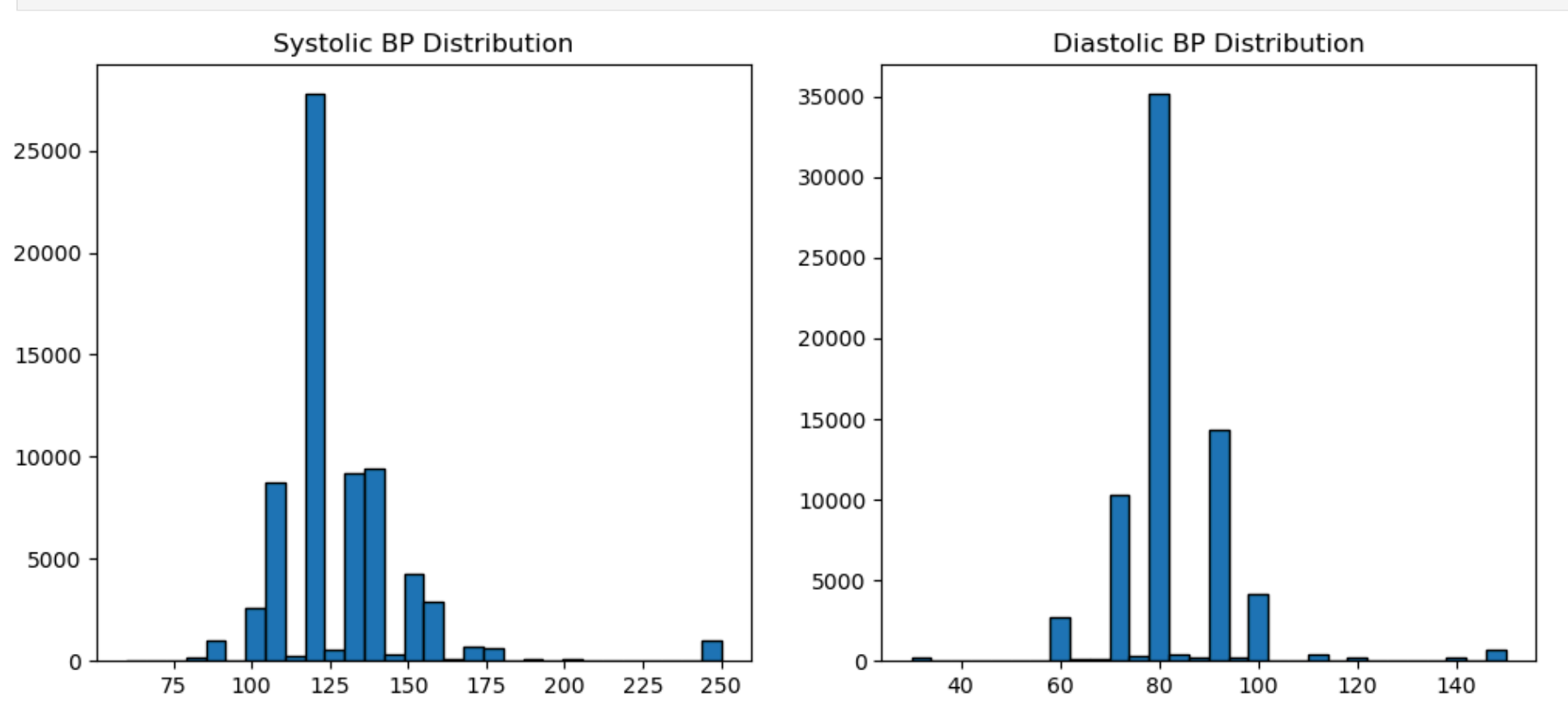
```
plt.figure(figsize=(6,4))
sns.countplot(x="active", data=df)
plt.title("Active Lifestyle")
plt.show()
```



In [39]:

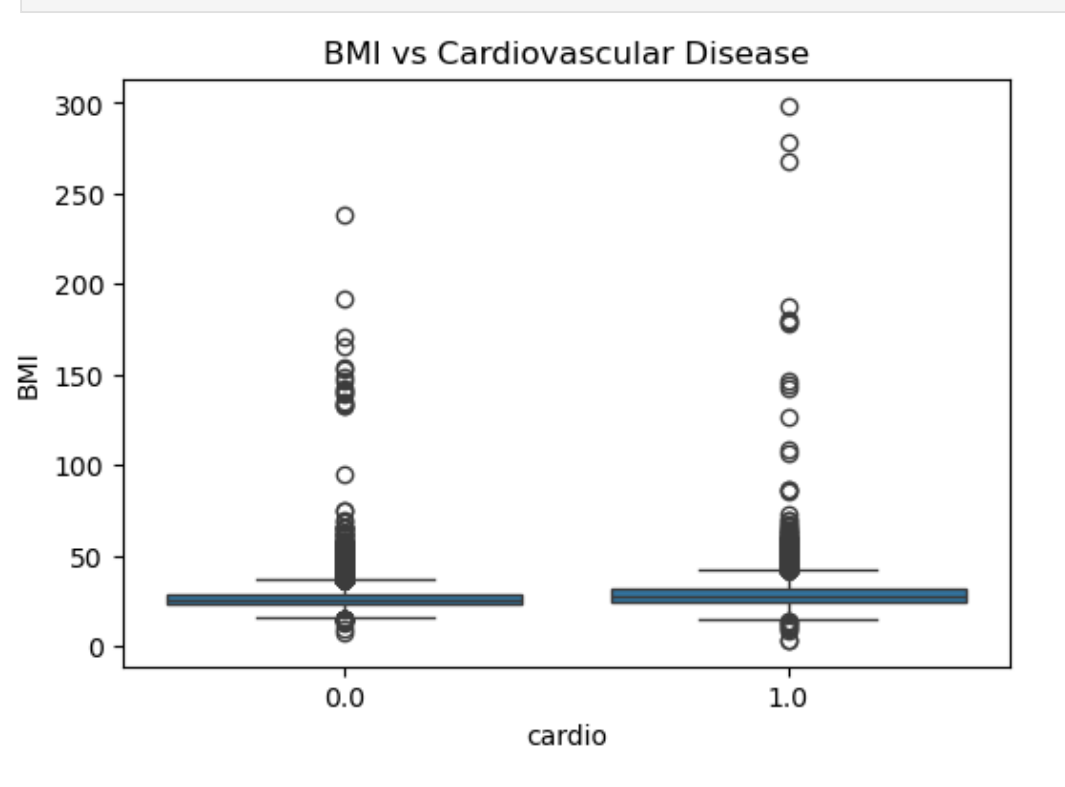
```
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.hist(df["ap_hi"], bins=30, edgecolor="black")
plt.title("Systolic BP Distribution")

plt.subplot(1,2,2)
plt.hist(df["ap_lo"], bins=30, edgecolor="black")
plt.title("Diastolic BP Distribution")
plt.show()
```



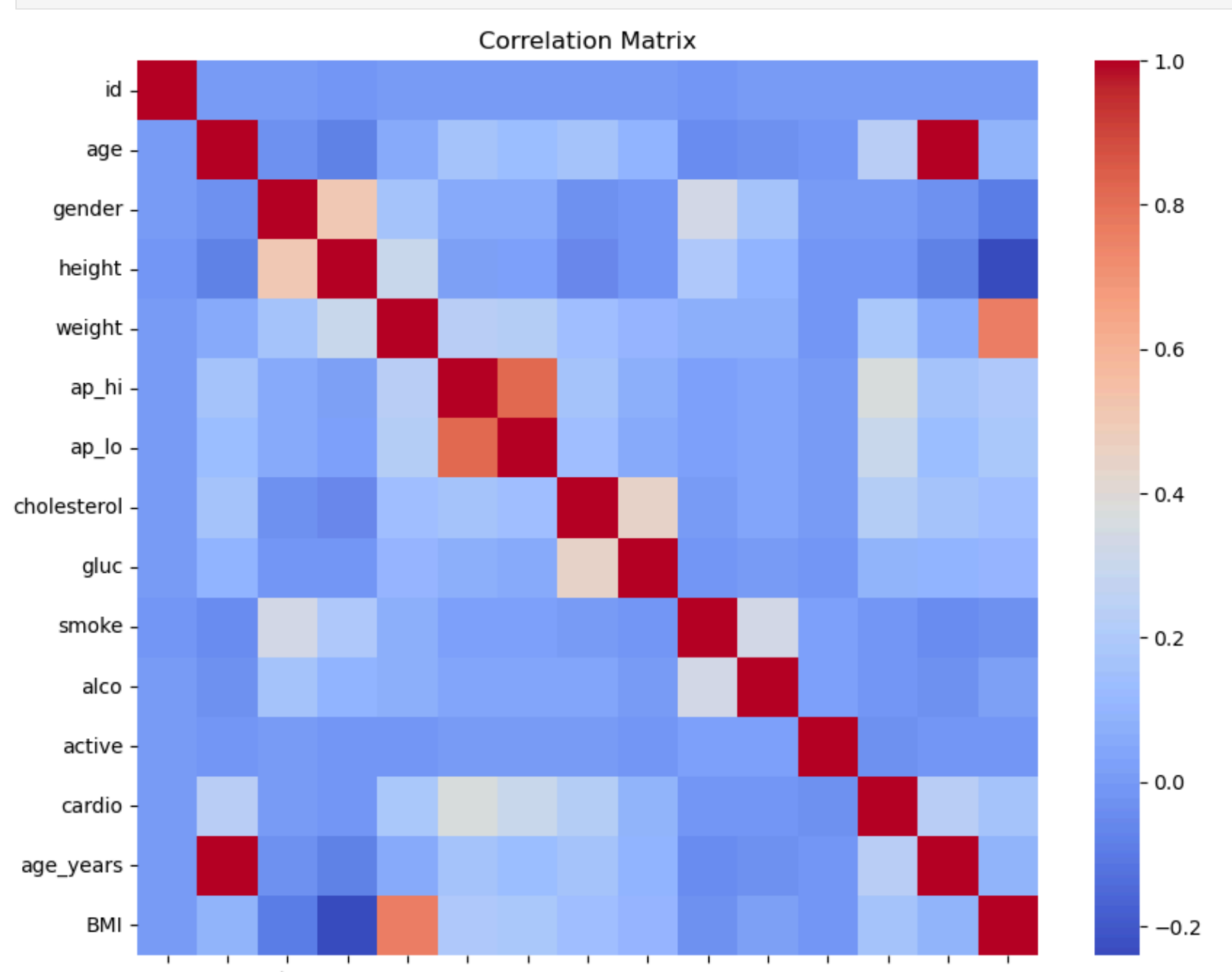
In [40]:

```
plt.figure(figsize=(6,4))
sns.boxplot(x="cardio", y="BMI", data=df)
plt.title("BMI vs Cardiovascular Disease")
plt.show()
```



In [41]:

```
corr = df.corr()
plt.figure(figsize=(10,8))
sns.heatmap(corr, cmap="coolwarm", annot=False)
plt.title("Correlation Matrix")
plt.show()
```



In [42]:

```
X = df.drop(columns=["cardio","id"])
y = df["cardio"]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

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

In [43]:

```
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train, y_train)
acc_lr = accuracy_score(y_test, log_reg.predict(X_test))
print("Logistic Regression Accuracy:", acc_lr)
```

```
Logistic Regression Accuracy: 0.7216428571428571
```

In [46]:

```
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)
acc_dt = accuracy_score(y_test, dt.predict(X_test))
print("Decision Tree Accuracy:", acc_dt)
```

```
Decision Tree Accuracy: 0.6305714285714286
```

In [20]:

```
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)
acc_rf = accuracy_score(y_test, rf.predict(X_test))
print("Random Forest Accuracy:", acc_rf)
```

```
Random Forest Accuracy: 0.7132142857142857
```

In [28]:

```
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
acc_knn = accuracy_score(y_test, knn.predict(X_test))
print("KNN Accuracy:", acc_knn)
```

```
KNN Accuracy: 0.6965
```

In [45]:

```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

svm = SVC(kernel="linear", cache_size=1000)

svm.fit(X_train_scaled, y_train)

y_pred_svm = svm.predict(X_test_scaled)

acc_svm = accuracy_score(y_test, y_pred_svm)
print("SVM Accuracy:", round(acc_svm*100, 2), "%")
```

```
SVM Accuracy: 72.39 %
```

In [54]:

```
X2 = df.drop(["id", "cardio"], axis=1)
y2 = df["cardio"]

X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_size=0.2, random_state=42, stratify=y2)

scaler2 = StandardScaler()
X2_train = scaler2.fit_transform(X2_train)
X2_test = scaler2.transform(X2_test)

best_model.fit(X2_train, y2_train)

print("Best model retrained on 12 features (without 'id').")
```

```
Best model retrained on 12 features (without 'id').
```

In [ ]:

```
X2 = df.drop(["id", "cardio"], axis=1).values
y2 = df["cardio"].values

X2_train, X2_test, y2_train, y2_test = train_test_split(
    X2, y2, test_size=0.2, random_state=42, stratify=y2
)

scaler2 = StandardScaler()
X2_train = scaler2.fit_transform(X2_train)
X2_test = scaler2.transform(X2_test)

accuracies = {
    "Logistic Regression": acc_lr,
    "Decision Tree": acc_dt,
    "Random Forest": acc_rf,
    "KNN": acc_knn,
    "SVM": acc_svm
}

best_model_name = max(accuracies, key=accuracies.get)
print("\n Best Model is:", best_model_name, "with accuracy:", accuracies[best_model_name])

if best_model_name == "Logistic Regression":
    best_model = log_reg
elif best_model_name == "Decision Tree":
    best_model = dt
elif best_model_name == "Random Forest":
    best_model = rf
elif best_model_name == "KNN":
    best_model = knn
else:
    best_model = svm

best_model.fit(X2_train, y2_train)

joblib.dump(best_model, "best_heart_model.joblib")

new_patient = np.array([[45, 1, 165, 70, 130, 85, 1, 1, 0, 0, 1, 25.7]])

new_patient_scaled = scaler2.transform(new_patient)

prediction = best_model.predict(new_patient_scaled)

if prediction[0] == 1:
    print("Prediction: Patient is at risk of Cardiovascular Disease")
else:
    print("Prediction: Patient is Healthy")
```

```
Best Model is: DTR with accuracy: 0.7529717142857143
```