In [1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [3]:
```
df1= pd.read_csv(r"C:\Users\bhava\Downloads\cardio_train.csv", delimiter=';')
df1
```

Out[3]:

	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...
69995	99993	19240	2	168	76.0	120	80	1	1	1	0	1	0
69996	99995	22601	1	158	126.0	140	90	2	2	0	0	1	1
69997	99996	19066	2	183	105.0	180	90	3	1	0	1	0	1
69998	99998	22431	1	163	72.0	135	80	1	2	0	0	0	1
69999	99999	20540	1	170	72.0	120	80	2	1	0	0	1	0

70000 rows × 13 columns

In [5]:
```
df1['age_years'] = df1['age'] // 365
df1['age_years']
```

Out[5]:
```
0        50
1        55
2        51
3        48
4        47
         ..
69995    52
69996    61
69997    52
69998    61
69999    56
Name: age_years, Length: 70000, dtype: int64
```

In [9]:
```
X = df1[['age', 'height', 'weight', 'ap_hi', 'ap_lo', 'cholesterol', 'gluc', 'smoke', 'alco', 'active']]
y = df1['cardio']
X
y
```

Out[9]:
```
0        0
1        1
2        1
3        1
4        0
        ..
69995    0
69996    1
69997    1
69998    1
69999    0
Name: cardio, Length: 70000, dtype: int64
```

In [11]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train, X_test, y_train, y_test
```

Out[11]:
```
(         age  height  weight  ap_hi  ap_lo  cholesterol  gluc  smoke  alco  \
47339  21876     154    80.0    130     90            2     1      0     0
67456  16717     162    70.0    140     90            1     1      0     0
12308  21128     174    92.0    150    100            1     1      0     0
32557  23366     173    76.0    120     82            1     1      0     0
664    20281     160    60.0    120     80            1     1      0     0
...      ...     ...     ...    ...    ...          ...   ...    ...   ...
37194  16001     170    75.0    150     80            1     1      1     0
6265   23209     162    73.0    160     90            1     1      0     0
54886  23589     169    74.0    120     80            1     1      0     0
860    18227     167    70.0    120     80            1     1      0     0
15795  15114     177    64.0    120     80            1     1      0     0

       active
47339       1
67456       0
12308       1
32557       1
664         1
...       ...
37194       1
6265        1
54886       1
860         0
15795       1

[56000 rows x 10 columns],
         age  height  weight  ap_hi  ap_lo  cholesterol  gluc  smoke  alco  \
46730  21770     156    64.0    140     80            2     1      0     0
48393  21876     170    85.0    160     90            1     1      0     0
41416  23270     151    90.0    130     80            1     1      0     0
34506  19741     159    97.0    120     80            1     1      0     0
43725  18395     164    68.0    120     80            1     1      0     0
...      ...     ...     ...    ...    ...          ...   ...    ...   ...
21525  20490     172    70.0    120     80            1     1      0     0
16276  16797     174    96.0    120     80            1     2      0     0
24390  22607     165    66.0    110     80            1     1      0     0
28061  19670     157    89.0    120     80            3     3      0     0
63452  18320     176    83.0    140     90            1     1      1     0

       active
46730       1
48393       1
41416       1
34506       1
43725       1
...       ...
21525       0
16276       1
24390       0
28061       1
63452       0

[14000 rows x 10 columns],
47339    1
67456    1
12308    1
32557    1
664      0
        ..
37194    1
6265     1
54886    0
860      0
15795    0
Name: cardio, Length: 56000, dtype: int64,
46730    1
48393    1
41416    1
34506    1
43725    0
        ..
21525    1
16276    1
24390    0
28061    1
63452    1
Name: cardio, Length: 14000, dtype: int64)
```

In [13]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
X_test
```

Out[13]:
```
array([[ 0.93346072, -1.01963223, -0.7094615 , ..., -0.31207962,
        -0.24087574,  0.49410558],
       [ 0.97638644,  0.69031476,  0.75283082, ..., -0.31207962,
        -0.24087574,  0.49410558],
       [ 1.54090021, -1.63032759,  1.10099566, ..., -0.31207962,
        -0.24087574,  0.49410558],
       ...,
       [ 1.27241195,  0.07961941, -0.57019556, ..., -0.31207962,
        -0.24087574, -2.02385896],
       [ 0.08304544, -0.89749316,  1.03136269, ..., -0.31207962,
        -0.24087574,  0.49410558],
       [-0.4636501 ,  1.42314919,  0.61356489, ...,  3.20431048,
        -0.24087574, -2.02385896]])
```

In [15]:
```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

In [17]:
```
svm = SVC()
svm.fit(X_train, y_train)
```

Out[17]:
```
▾ SVC

SVC()
```

In [18]:
```
y_pred_svm = svm.predict(X_test)
y_pred_svm
```

Out[18]:
```
array([1, 1, 1, ..., 1, 1, 1], dtype=int64)
```

In [21]:
```
accuracy_svm = accuracy_score(y_test, y_pred_svm)
print(f'Accuracy of SVM: {accuracy_svm * 100:.2f}%')
```

```
Accuracy of SVM: 73.11%
```

In [22]:
```
# Import the KNN classifier
from sklearn.neighbors import KNeighborsClassifier

# Step 1: Create and train the KNN model
knn = KNeighborsClassifier(n_neighbors=5)  # You can change 'n_neighbors' to optimize performance
knn.fit(X_train, y_train)

# Step 2: Make predictions
y_pred_knn = knn.predict(X_test)

# Step 3: Calculate the accuracy
accuracy_knn = accuracy_score(y_test, y_pred_knn)
print(f'Accuracy of KNN: {accuracy_knn * 100:.2f}%')
```

```
Accuracy of KNN: 65.64%
```

In [23]:
```
# Import the Decision Tree classifier
from sklearn.tree import DecisionTreeClassifier

# Step 1: Create and train the Decision Tree model
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)

# Step 2: Make predictions
y_pred_dt = dt.predict(X_test)

# Step 3: Calculate the accuracy
accuracy_dt = accuracy_score(y_test, y_pred_dt)
print(f'Accuracy of Decision Tree: {accuracy_dt * 100:.2f}%')
```

```
Accuracy of Decision Tree: 63.35%
```

In [24]:
```
# Import the Logistic Regression classifier
from sklearn.linear_model import LogisticRegression

# Step 1: Create and train the Logistic Regression model
lr = LogisticRegression(random_state=42)
lr.fit(X_train, y_train)

# Step 2: Make predictions
y_pred_lr = lr.predict(X_test)

# Step 3: Calculate the accuracy
accuracy_lr = accuracy_score(y_test, y_pred_lr)
print(f'Accuracy of Logistic Regression: {accuracy_lr * 100:.2f}%')
```

```
Accuracy of Logistic Regression: 72.38%
```

In [25]:
```
# Import the Random Forest classifier
```

```
from sklearn.ensemble import RandomForestClassifier

# Step 1: Create and train the Random Forest model
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)

# Step 2: Make predictions
y_pred_rf = rf.predict(X_test)

# Step 3: Calculate the accuracy
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f'Accuracy of Random Forest: {accuracy_rf * 100:.2f}%')
```

```
Accuracy of Random Forest: 71.44%
```

In [26]:

```
# Summarizing the accuracy of all models
print(f"Support Vector Machine Accuracy: {accuracy_svm * 100:.2f}%")
print(f"K-Nearest Neighbor Accuracy: {accuracy_knn * 100:.2f}%")
print(f"Decision Tree Accuracy: {accuracy_dt * 100:.2f}%")
print(f"Logistic Regression Accuracy: {accuracy_lr * 100:.2f}%")
print(f"Random Forest Accuracy: {accuracy_rf * 100:.2f}%")
```

```
Support Vector Machine Accuracy: 73.11%
K-Nearest Neighbor Accuracy: 65.64%
Decision Tree Accuracy: 63.35%
Logistic Regression Accuracy: 72.38%
Random Forest Accuracy: 71.44%
```