Cardiovascular disease dataset analysis using different Machine learning algorithms

Introduction:

The objective of this Analysis is to predict weather a person is suffering from cardiovascular disease using machine learning algorithms. In this analysis I am comparing some of the classification algorithms like Decision Tree, Random Forest, Support vector machine, KNN etc., to find out the best model and the algorithm to predict the cardiovascular disease and the key features which are impacting it like Blood pressure (Systolic, Diastolic Blood pressure), Glucose, smoking, Activity, Age etc.

The dataset which I am using is from Kaggle (Machine Learning and Data Science Community). The values in that dataset are collected at the moment of the medical examination.

Description of the dataset:

The dataset contains total of 70000 rows and 13 columns in total and the column names and description are as follows:

Column name	Description	Feature	Values /data type
age	Age of the person	Objective feature	Int (Days)
height	Height of the person	Objective feature	int (cm)
weight	Weight of the person	Objective feature	Float (Kg)
gender	Gender of the person	Categorical code	0 – female 1- male
ap_hi	Systolic blood pressure	Examination feature	int
ap_lo	Diastolic blood pressure	Examination feature	int
Cholesterol	The amount of the cholesterol the person is having	Examination feature	1: normal 2: above normal 3: well – above normal
gluc	The amount of the glucose present in the body of that person	Examination feature	1: normal 2: above normal 3: well above normal
smoke	Whether the person is smoking or not	Subjective feature (binary)	0 – non-smoker 1 - smoker
alco	Whether the person drinks alcohol or not	Subjective feature (binary	0-non-alchoholic 1- alcoholic
active	Whether the person is physically active	Subjective feature (binary)	0 – not- active 1-active

cardio	Presence or absence of cardiovascular disease	Target variable (binary)	0- No 1- Yes

The above is the tabular column of the columns containing in the dataset. There are 3 types of input features given in the dataset these are as follows:

Objective: factual information.
Examination: results of medical Examination.
Subjective: Information given by the patient.
Target variable (cardio): person is having cardiovascular disease or not.

Exploratory data Analysis (EDA):

Before implementing the model, we need to preprocess the data in order to remove any outliers present in the data and need to check for any missing values or null values and mismatched data.

As per the given data the data is having all the numerical variables even there are some categorical variables i.e., (smoke, gender, alcohol, glucose, physical activity) but they are also given as numerical values i.e., binary values 0 and 1 including the target variable (cardio).

Upon checking for the missing values, **the dataset does not have any missing values** to deal with, the dataset is good and having the efficient data.

Comparing the percentage of people who is suffering with cardiovascular disease and are not suffering with it. The percentages are as follows.

Cardiovascular disease	**percentage**
Yes – people having the disease	49.97%
No – people without having the disease	50.03%

Outliers in the given data:

Blood pressure:

Based on the columns in the dataset the ap_hi, ap_lo columns are related to blood pressure i.e., Systolic Blood pressure and Diastolic Blood pressure these are most related for the cause of Cardio vascular disease.

The normal Blood pressure of a healthy person is (120/80) 120- Systolic Blood pressure and 80(Diastolic Blood pressure), According to the Medical researchers and Doctors the Minimum and maximum values of this blood pressure are,

Systolic blood pressure – Minimum = 0, maximum = 300mm Hg
Diastolic blood pressure – Minimum = 0, Maximum = 200mmHg

Note: Always the Systolic Blood pressure must be higher than that of the Diastolic blood pressure.
Based on these conditions we need to clean the data in order to eliminate the outliers in the dataset and the percentage of this outlier values is 1.84%, we can remove these values.
BMI: The body mass index is the measure of Fat based on the persons height and weight and it will play a crucial role in the cardio vascular disease if a person with more BMI will have more chances to get the Cardio vascular disease. The formula of the BMI is

BMI = Weight (in Kgs)/ (Height * Height) (in mts)
In the Given Data we have both height and weight columns so we can obtain the BMI value based on these 2 columns and compare with the target variable (Cardio).
Underweight: BMI < 18.5
Normal weight: 18.5 <= BMI < 25
Overweight: 25 <= BMI < 30
Obesity class I: 30 <= BMI < 35
Obesity class II: 35 <= BMI < 40
Obesity class III: BMI >= 40
Based on these values we can give the Minimum and Maximum value ranges for BMI and remove the values which are not in that particular range.
Upon checking the total percentage of outlier values related to BMI is 2.83%, we can remove these values from the dataset.
Age: Age is also a concerning factor for the cardio vascular disease upon increase in age. The increase in age will automatically increase the chances of getting the Cardio vascular disease and so according to the data the age is given in days we can convert it to years and check the correlation between age and cardio (target variable).
Gender: In the dataset the gender is classified as Male/Female gender 1- female and 2: Male. On checking the value counts for this column Most the data is for the female and only 30% of the data is Male.
Other columns:
The other columns are usually categorical columns but, in the dataset, they have directly given as the numerical/binary values and we can classify accordingly.
Usually the other columns Cholesterol, glucose level, alcohol intake, smoke (Number of cigs per day), Physical Activity these columns also will play a crucial role in the cardio vascular disease but upon checking the value counts data the Data in the Data set is mostly Skewed data i.e., it is having 80 to 20% ratio between each column so that for this particular dataset these columns are not that much concerning columns for the Cardio vascular disease.

Methods:

For Analysis and predicting of the Heart stroke (Cardio vascular disease) I am using 4 different algorithms i.e.,

1. Decision tree
2. Random Forest
3. K-Nearest Neighbor
4. SVM (Support vector Machine)

Upon checking the Accuracy and sensitivity of the models we can able to tell that which model is better in order to predict the cardio vascular disease with the given Data.

For the model I have used the normal test train split ratio i.e., 80:20 for splitting the data to training and test data

Metrics used throughout the models are:

Accuracy: Accuracy is defined as the measure how well the model is the able to predict the value of the given data sample. Accuracy can be calculated as Total number of true predictions/Total number of predictions made.

Sensitivity: Sensitivity is the measure of the model that is able to correctly identify the True outcomes from a prediction.

Sensitivity can be calculated as follows:

Sensitivity = T.P/ (T. P+F.N)

T.P: Ture positive

F.N: False Negative

Decision Tree:

The decision tree algorithm is a powerful tool for solving both classification and regression problems. This algorithm constructs a tree-like model by recursively partitioning the dataset based on different conditions. In the context of predicting the likelihood of a heart stroke for an individual, the decision tree algorithm can use the available data to make a prediction.

Implementation:

1. The basic decision tree algorithm on the dataset gives without passing any parameters to the Decision tree classifier and checking the predicted value i.e., y_pred. The Accuracy and sensitivity for the given data is as follows:

   Sensitivity = 0.62333% i.e., 62.4%

   Accuracy = 0.64% i.e., 64%

Here in the above model, we did not used any parameters which can increase the accuracy and sensitivity. The Accuracy by using the Decision Tree algorithm can be changed based on the certain parameters like max_depth & max_leaf_nodes etc.,

Max_depth: In a decision tree algorithm, the depth is the distance from the root node to the farthest leaf node. By setting the max_depth parameter, we can control the complexity of the decision tree and improve the accuracy of the model. The max_depth parameter limits the number of levels in the tree, and we can optimize this parameter through hyperparameter tuning. By selecting the max_depth value that yields the highest validation set accuracy, we can build a more accurate decision tree model.

Max_leaf_nodes: A leaf node in a decision tree algorithm is a node that has no children and represents a final prediction based on the given input data. By setting the max_leaf_nodes parameter, we can avoid overfitting by constraining the complexity of the decision tree. This parameter limits the number of leaf nodes in the tree and can be optimized through hyperparameter tuning. By selecting an appropriate value for max_leaf_nodes, we can prevent the decision tree from becoming too complex and overfitting the training data, resulting in a more generalizable model.

Hyperparameter Tuning for both Max_depth and Max_leaf_nodes:

Hyperparameter tuning is a method to obtain the optimal values for the hyper parameters of a machine learning algorithm to achieve the best performance on a validation set.

On performing the Hyperparameter tuning on the above model we are able to get the best parameter values for the max_depth & max_leaf_nodes

Max_depth = 5

Max_leaf_nodes = 25

The accuracy of the Decision tree classifier by using this Tuning technique and using the above parameters is:

Sensitivity: 64%

Accuracy:0.73 i.e., 73%

Random forest:

The Random Forest algorithm is a type of ensemble learning technique that builds numerous decision trees during training and generates predictions by taking the mode of the classes assigned by the individual trees in case of classification problems.

Implementation:

1. Basic random forest algorithm is implemented by passing the parameter values randomly the accuracy and sensitivity values are:
   Accuracy: 68.4%
   Sensitivity: 69.05%

For the basic model this is the maximum accuracy and sensitivity we obtained.

Hyper parameter tuning for n_estimators & max_features, max_depth and max_leaf_nodes:

Tuned random forest: I have tuned the random forest algorithm based for different parameters using the GridSearchCV randomizer.

n_estimators: It is a hyper parameter in random forest classifier which represents the number of decision trees to be used in the ensemble. The greater number of estimators the more will be the accuracy.

The best features for this I have obtained is as follows:

max_depth :20, max_leaf_nodes: 1000, n_estimators = 500, max_features: ‘sqrt’
On tuning the model, we got the best accuracy for the random forest.
Accuracy: 73.04%
Sensitivity:69.05%

Note: Before implementing the KNN and SVM algorithm models I have used standard scaler method in order to standardize the train and test data.

Standardization. The process of standardization involves adjusting the scale of features in a dataset to achieve a mean of zero and a standard deviation of one. This technique is often applied as a preprocessing step to improve the data's compatibility with analysis and modeling techniques, especially when using algorithms that are sensitive to the feature scale, like k-nearest neighbors, SVM, and logistic regression.
K-Nearest Neighbor:
KNN is a Non parametric supervised learning algorithm the input consists of K closest training examples in the feature
The choice of K is the hyperparameter that affect the accuracy of the model the best value for K will give the most accuracy for the model
Implementation:

1. Basic model for KNN with a variety value of k in ranging from (1,50) and the accuracy is almost same after the K value = 21.
2. For k =21 the model has predicted the below accuracy and sensitivity values.
   Accuracy: 73%
   Sensitivity: 70%

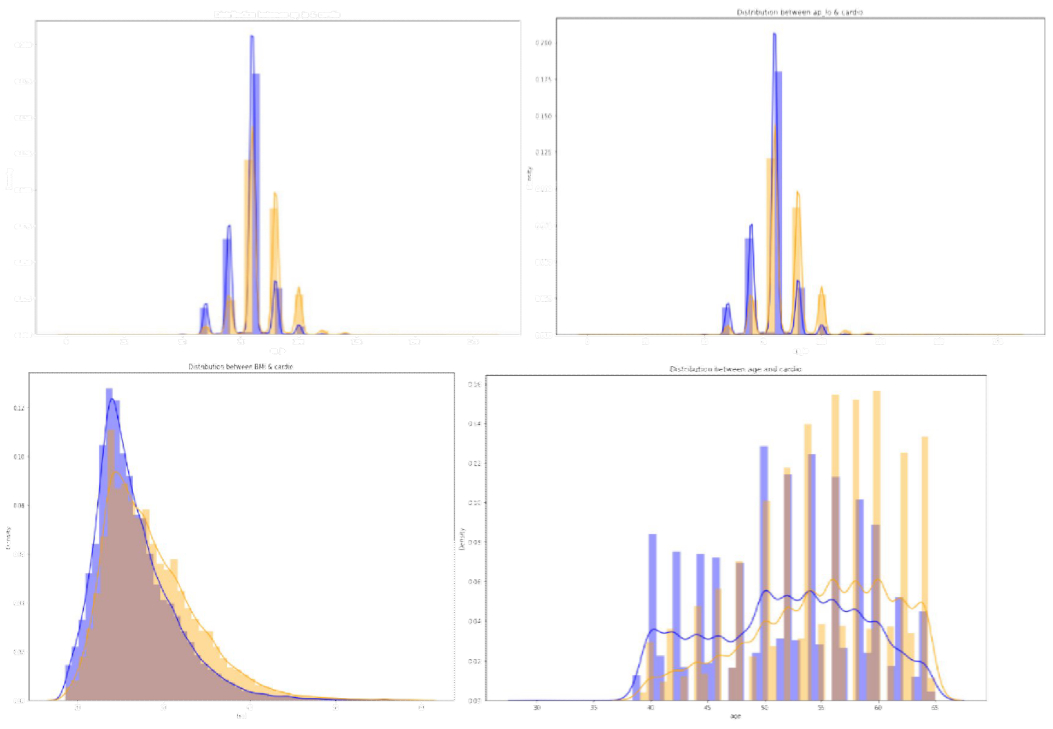
Tuning the KNN for best hyperparameters:
Using GridSearchCV to obtain the best values for the parameters n_neighbours, weights, Metric.
The best parameters obtained after tuning the ML- model are
Metric: ‘manhattan, n_neighbors: 35, weights: ’uniform’
After using these values also, the sensitivity has changed but the accuracy remained almost similar to that of the normal KNN model.
Support vector machine:
Implemented the basic model of the SVM. By using kernel parameter to ‘rbf’.
The sensitivity and the accuracy of the model is as follows:
Accuracy: 0.735
Sensitivity :0.6768
The basic model itself have the high accuracy upon implementing the standard scaler method on train and test datasets.
Results:
1.Based on the above ML models created I have compared all the models with each other for both accuracy and sensitivity
2.The accuracy is same almost in every model but Random Forest and Support vector machine has somewhat better accuracy compared to other models.

3.The sensitivity is different for each model so based on these results the better performing Models are **Random Forest and Support vector machine (SVM)**
The important features concerning the risk of cardiovascular disease are:
ap_hi (systolic blood pressure)
ap_lo (Diastolic blood pressure)
Age (Age of the person)
BMI (Body mass Index)

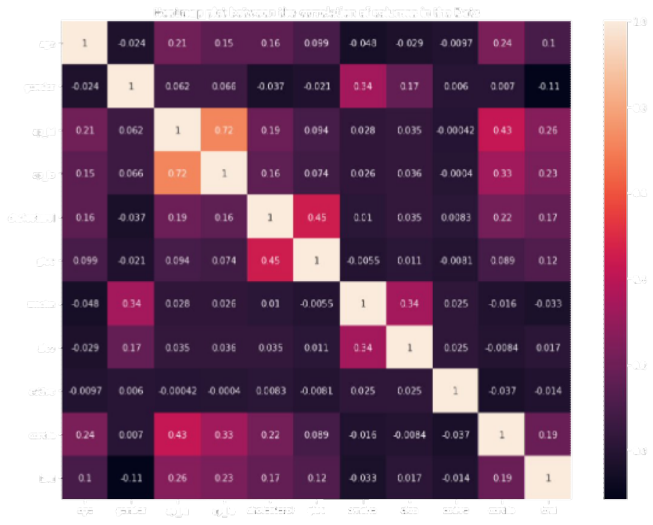
Below is the table Comparison of Accuracy and Sensitivity:

Id	Ml-model	Accuracy	Sensitivity
1	Decision tree	0.732505	0.633078
2	Random forest	0.734049	0.690508
3	K-NearestNeighbor	0.726110	0.695307
4	Svm (Support vector machine)	0.735225	0.676863

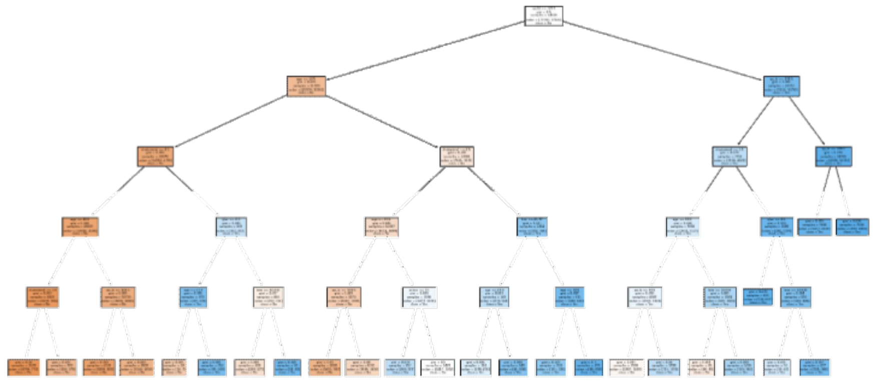
Figures: 1. Plots related to EDA target variable (cardio vs ap_hi), (cardio vs ap_lo), (cardio vs bmi) & (cardio vs age): the plots are distribution plots showing how the target variable is changing with these columns.



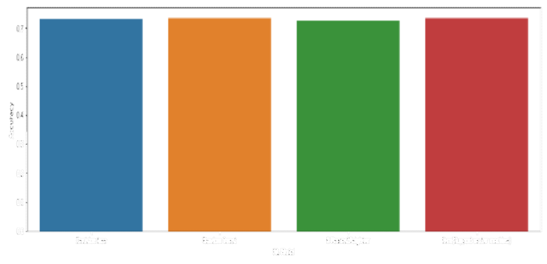
Correlation between the columns: A heat map showing the correlation between the variables.

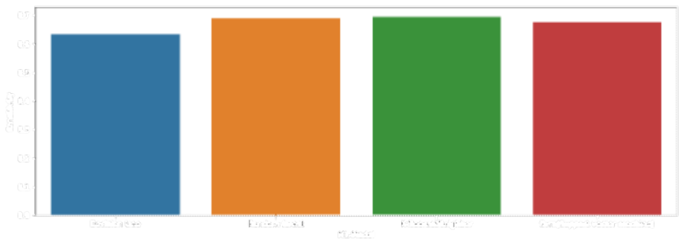


Decision tree model graph(tuned):



Accuracy and sensitivity with 4 algorithms





Conclusion:

Based on the results and the comparison between the 4 Ml -models on the cardio vascular disease dataset, The random forest and the Support vector machine are performing well and the features of importance are identified.

1. Importing all the necesssary libraries required

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn import svm
from sklearn import metrics
```

In [2]:
```
from IPython.display import display
# Reading the dataset extracted from the Kaggle
cardio_disease = pd.read_csv("cardio_train.csv" , delimiter = ';')

# viewing the head and tail of the cardio disease dataset #
display(cardio_disease.head(5))
cardio_disease.tail(5)
```

	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

Out[2]:

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active
69995	99993	19240	2	168	76.0	120	80	1	1	1	0	1
69996	99995	22601	1	158	126.0	140	90	2	2	0	0	1
69997	99996	19066	2	183	105.0	180	90	3	1	0	1	0
69998	99998	22431	1	163	72.0	135	80	1	2	0	0	0
69999	99999	20540	1	170	72.0	120	80	2	1	0	0	1

In [3]:
```
# info of the dataset
cardio_disease.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70000 entries, 0 to 69999
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
```

```
 0   id           70000 non-null  int64
 1   age          70000 non-null  int64
 2   gender       70000 non-null  int64
 3   height       70000 non-null  int64
 4   weight       70000 non-null  float64
 5   ap_hi        70000 non-null  int64
 6   ap_lo        70000 non-null  int64
 7   cholesterol  70000 non-null  int64
 8   gluc         70000 non-null  int64
 9   smoke        70000 non-null  int64
 10  alco         70000 non-null  int64
 11  active       70000 non-null  int64
 12  cardio       70000 non-null  int64
dtypes: float64(1), int64(12)
memory usage: 6.9 MB
```

col_description	feature	col_name	datatype

Age | Objective Feature | age | int (days) Height | Objective Feature | height | int (cm) | Weight | Objective Feature | weight | float (kg) | Gender | Objective Feature | gender | categorical code | Systolic blood pressure | Examination Feature | ap_hi | int | Diastolic blood pressure | Examination Feature | ap_lo | int | Cholesterol | Examination Feature | cholesterol | 1: normal, 2: above normal, 3: well above normal | Glucose | Examination Feature | gluc | 1: normal, 2: above normal, 3: well above normal | Smoking | Subjective Feature | smoke | binary | Alcohol intake | Subjective Feature | alco | binary | Physical activity | Subjective Feature | active | binary | Presence or absence of cardiovascular disease | Target Variable | cardio | binary |

All the binary values (0, 1) are considered as:

Yes : 1

No : 0

Description of the column names of the cardio_disease dataset

Total number of rows it has is : 70000

Number of columns is : 13

Exploratory data Analysis

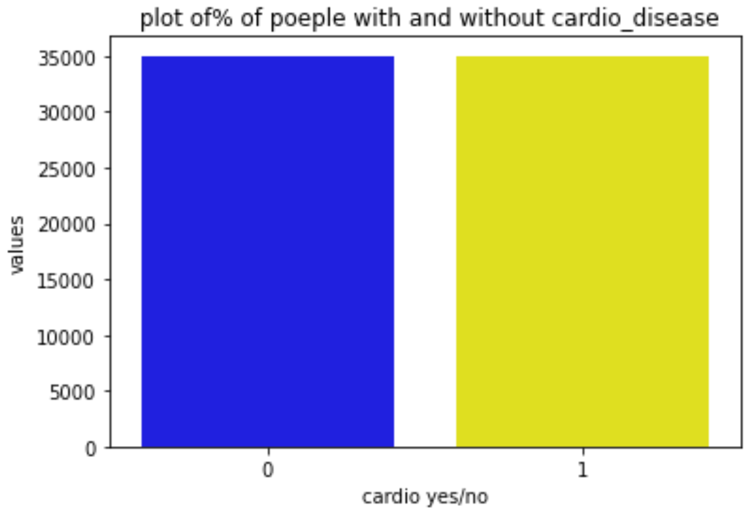
In [4]:

```python
# Checking how much % of the people are suffering with cardio disease and how many p
cardio_yes = sum(cardio_disease['cardio'] == 1)/ len(cardio_disease)*100
cardio_no  =  sum(cardio_disease['cardio'] == 0)/len(cardio_disease)*100
print("% of people having cardio disease:",cardio_yes)
print("% of people not having cardio disease:", cardio_no )
# visualizing the same
sns.countplot(data = cardio_disease, x = 'cardio', palette= {0 : 'blue', 1: 'yellow'
plt.title(" plot of% of poeple with and without cardio_disease")
plt.xlabel("cardio yes/no")
plt.ylabel("values")
plt.show
```

```
% of people having cardio disease: 49.97
% of people not having cardio disease: 50.029999999999994
```

Out[4]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [5]:

```
#checking for null values or missing values :
cardio_disease_columns = cardio_disease.columns.tolist()
print(cardio_disease_columns)
for col in cardio_disease_columns:
    i = 0
    for j in range(0, len(cardio_disease)):
        if pd.isnull(cardio_disease.iloc[j][col]) == True:
            i += 1
    print("Null values for", col, "is", i )
```

```
['id', 'age', 'gender', 'height', 'weight', 'ap_hi', 'ap_lo', 'cholesterol', 'gluc',
'smoke', 'alco', 'active', 'cardio']
Null values for id is 0
Null values for age is 0
Null values for gender is 0
Null values for height is 0
Null values for weight is 0
Null values for ap_hi is 0
Null values for ap_lo is 0
Null values for cholesterol is 0
Null values for gluc is 0
Null values for smoke is 0
Null values for alco is 0
Null values for active is 0
Null values for cardio is 0
```

Outlier checking:

Outlier checking for Blood pressure for ap_hi & ap_lo

According to the Doctors or medical reserachers in the cardiovascular diseases the chances of getting a cardiovascular disease is dependent on the Blood pressure i.e., both (Systolic and Diastolic blood pressure). according from google the minimum and maximum values for Systolic and Distolic are as follows

type	Minimum	Maximum
Systolic pressure	0	300 mm Hg
Diastolic pressure	0	200 mm Hg

Note: Systolic pressure will be always higher than diastolic pressure

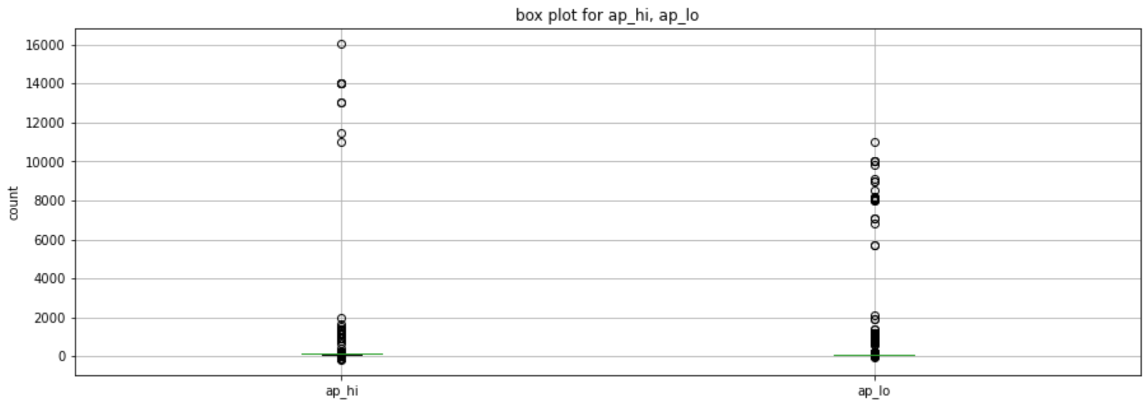
In [6]:

```
#box plot that shows the outliers for ap_hi & ap_lo
plt.figure(figsize = (15, 5))
```

```
cardio_disease.boxplot(['ap_hi' ,  'ap_lo'])
plt.title("box plot for ap_hi, ap_lo")
plt.ylabel("count")
plt.show
```

Out[6]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [7]:

```
#According to the above we can check for the outliers if any present in those both c
outlier_values_Bp = len(cardio_disease[(cardio_disease["ap_hi"] >= 300) | (cardio_di
print("The number of outlier values in Blood presuure values is :", outlier_values_B
percentage_of_outlier_values = np.round((outlier_values_Bp/len(cardio_disease)*100),
print("The percentage of outlier values in the dataset for both Blood pressures is:"
```

```
The number of outlier values in Blood presuure values is : 1289
The percentage of outlier values in the dataset for both Blood pressures is: 1.84
```

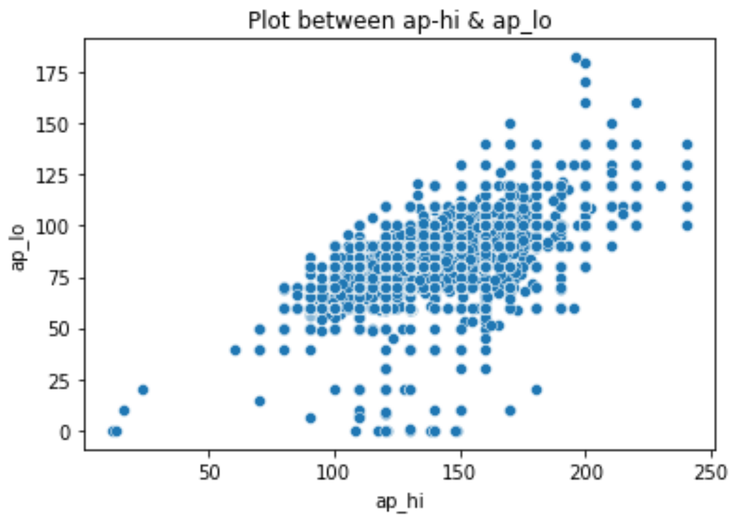
In [8]:

```
# since the % of outlier values is low we can directly remove them for the dataset #
cardio_disease_1 = cardio_disease.copy()
cardio_disease_1 = cardio_disease_1[(cardio_disease_1['ap_hi'] >= 0) & (cardio_disea
cardio_disease_1= cardio_disease_1[(cardio_disease_1['ap_hi'] <= 300) &(cardio_disea
cardio_disease_1 = cardio_disease_1[(cardio_disease_1['ap_hi'] > cardio_disease_1['a
cardio_disease_1.info()

# we have removed the outlier values from ap_hi, ap_lo
#plotting ap_hi vs ap_lo
sns.scatterplot(x = 'ap_hi', y = 'ap_lo', data = cardio_disease_1)
plt.title("Plot between ap-hi & ap_lo")
plt.xlabel("ap_hi")
plt.ylabel("ap_lo")
plt.show
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 68723 entries, 0 to 69999
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   id           68723 non-null  int64  
 1   age          68723 non-null  int64  
 2   gender       68723 non-null  int64  
 3   height       68723 non-null  int64  
 4   weight       68723 non-null  float64
 5   ap_hi        68723 non-null  int64  
 6   ap_lo        68723 non-null  int64  
 7   cholesterol  68723 non-null  int64  
 8   gluc         68723 non-null  int64  
 9   smoke        68723 non-null  int64  
 10  alco         68723 non-null  int64  
 11  active       68723 non-null  int64  
 12  cardio       68723 non-null  int64  
```

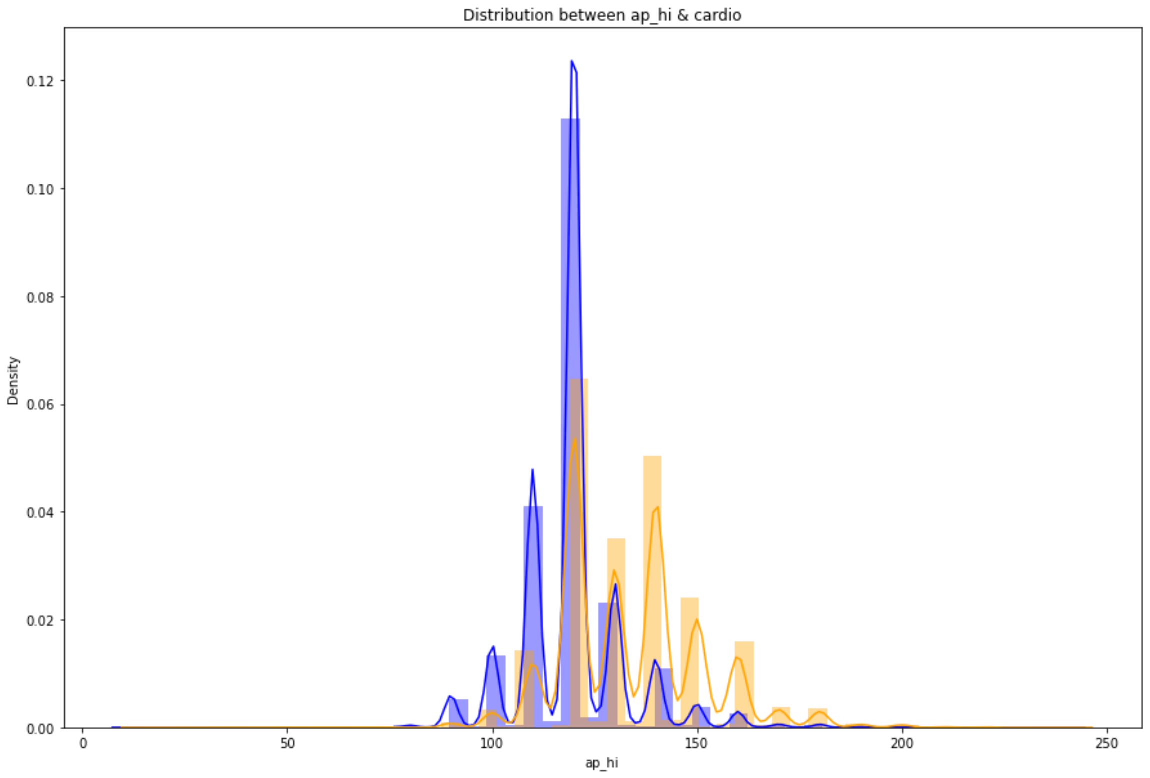
```
dtypes: float64(1), int64(12)
memory usage: 7.3 MB
```

Out[8]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [9]:
```
# distribution of the ap_hi according with the cardio disease
plt.figure(figsize = (15, 10))
sns.distplot(cardio_disease_1['ap_hi'][cardio_disease_1['cardio'] == 0], color = 'bl
sns.distplot(cardio_disease_1['ap_hi'][cardio_disease_1['cardio'] == 1], color = 'or
plt.title("Distribution between ap_hi & cardio")
plt.show()
```

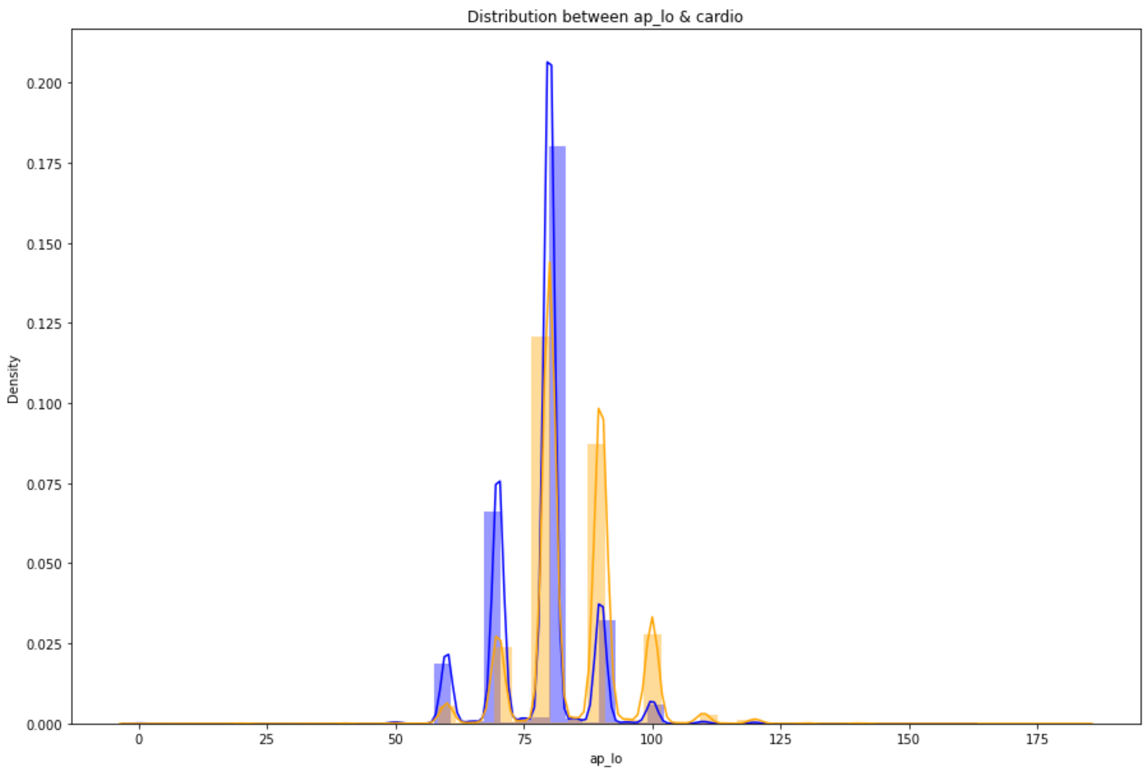
```
D:\Anaconda\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplo
t` is a deprecated function and will be removed in a future version. Please adapt yo
ur code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
D:\Anaconda\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplo
t` is a deprecated function and will be removed in a future version. Please adapt yo
ur code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



In [10]:

```
plt.figure(figsize = (15,10))
sns.distplot(cardio_disease_1['ap_lo'][cardio_disease_1['cardio'] == 0], color = 'bl
sns.distplot(cardio_disease_1['ap_lo'][cardio_disease_1['cardio'] == 1], color = 'or
plt.title("Distribution between ap_lo & cardio")
plt.show()
```

```
D:\Anaconda\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplo
t` is a deprecated function and will be removed in a future version. Please adapt yo
ur code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
D:\Anaconda\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplo
t` is a deprecated function and will be removed in a future version. Please adapt yo
ur code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



From the above graphs we can see that the maximum people affected with cardio or with abnormal values i.e range between min and maximum values.

height and weight outliers

we have both height and weight columns, with this we can calculate the BMI(Body mass Index) value which is a key factor affecting the cardio vascular disease.
The minimum and maximum value of bmi for a person likely to get Cardiovascular disease is 18.5 kg/sq.mts(underweight), 60 kg/sq.mts (overweight)

```
BMI = weight(kgs)/height**2
```

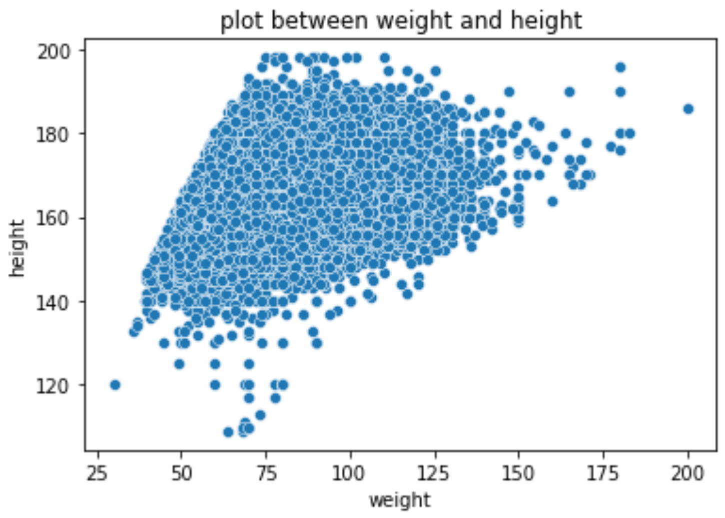
In [11]:
```
#calculating Bmi
cardio_disease_1['height']= cardio_disease_1['height'].astype(float)
cardio_disease_1['bmi'] = np.round(cardio_disease_1['weight']/((cardio_disease_1['he
cardio_disease_1['bmi'].head()
```

Out[11]:
```
0    21.97
1    34.93
2    23.51
3    28.71
4    23.01
Name: bmi, dtype: float64
```

In [12]:
```
# filtering the dataset and removing outliers based on the bmi max and min values:
cardio_disease_1 = cardio_disease_1[(cardio_disease_1['bmi'] < 60) & (cardio_disease
outlier_values_removed = round(70000 - len(cardio_disease_1))/70000*100
print("the total percentage of values  removed from the dataset:", outlier_values_re
```

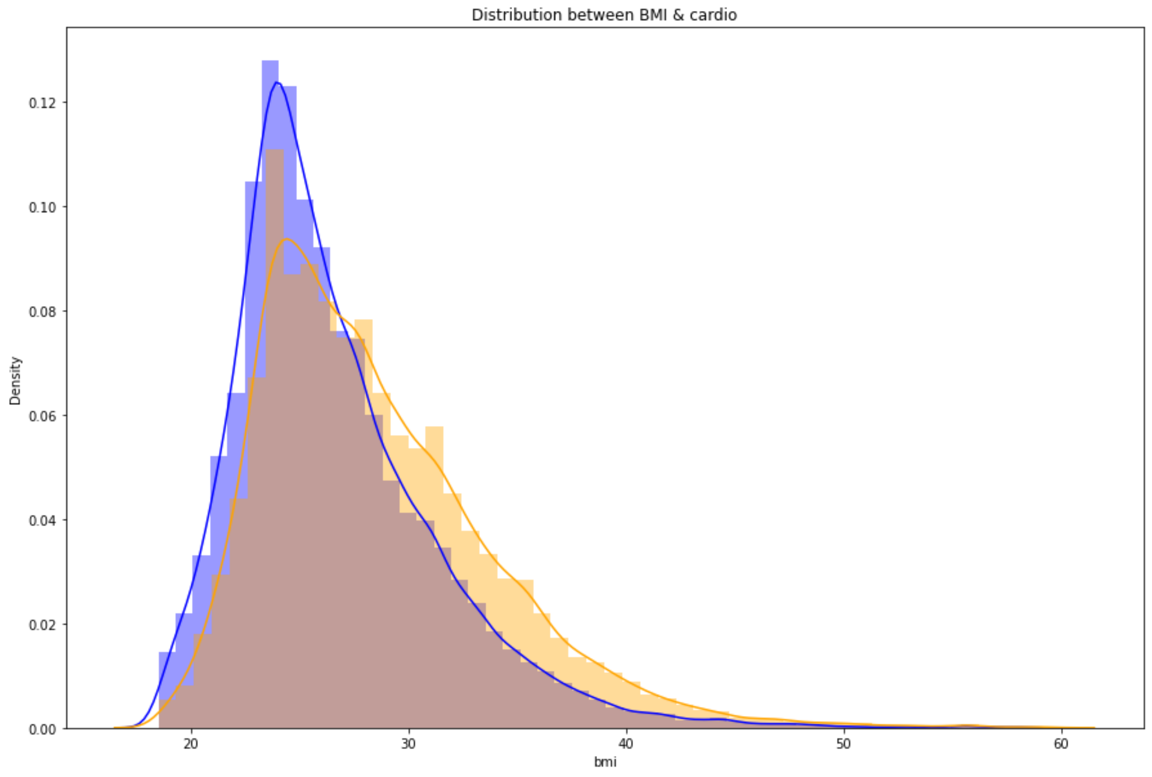
```
the total percentage of values  removed from the dataset: 2.8328571428571427
```

In [13]:
```
# plot between height and weight#
sns.scatterplot(x = 'weight', y = 'height', data = cardio_disease_1)
plt.title("plot between weight and height")
plt.show()
```



In [14]:
```
# distribution of BMI with cardio
plt.figure(figsize = (15, 10))
sns.distplot(cardio_disease_1['bmi'][cardio_disease_1['cardio'] == 0], color = 'blue
sns.distplot(cardio_disease_1['bmi'][cardio_disease_1['cardio'] == 1], color = 'oran
plt.title("Distribution between BMI & cardio")
plt.show()
```

```
D:\Anaconda\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplo
t` is a deprecated function and will be removed in a future version. Please adapt yo
ur code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
D:\Anaconda\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplo
t` is a deprecated function and will be removed in a future version. Please adapt yo
ur code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

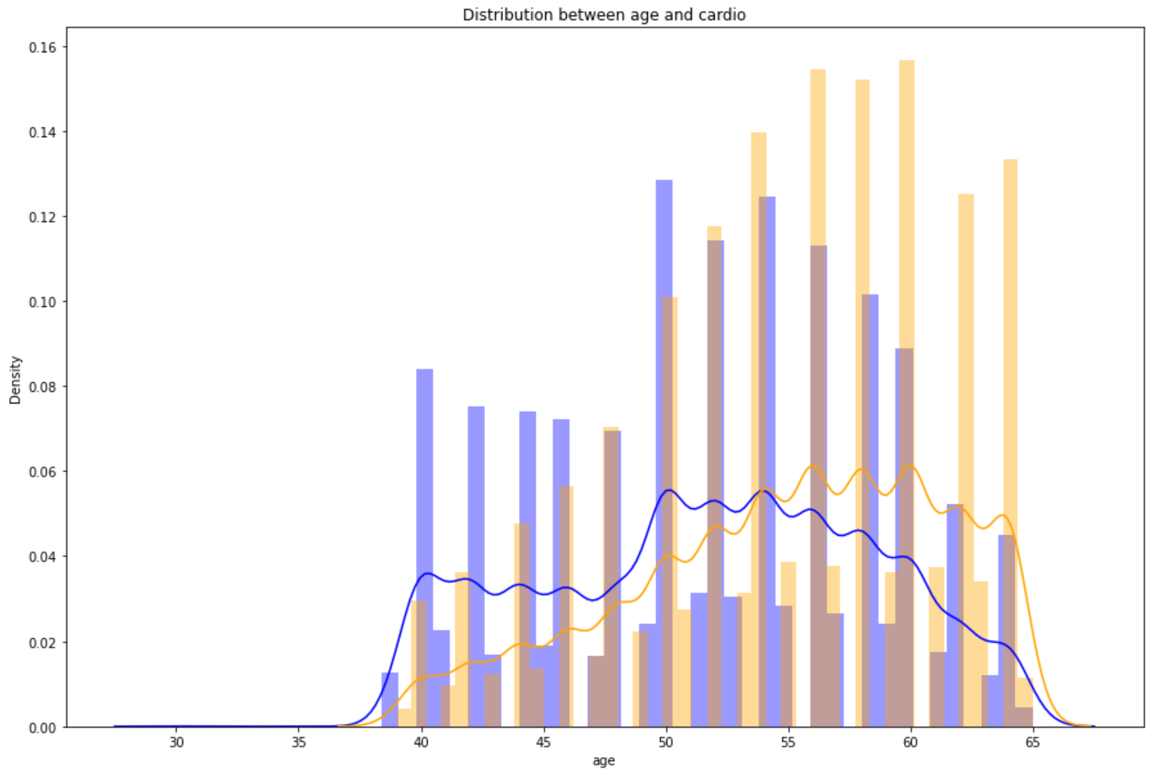


Age is one of the concerning factors for the cardio vascular disease.
It is said that as age increases the chances of getting a heat stroke(cardio vascular disease) will increase.

In [15]:
```
# In the data the Age is given in numbers converting the age into Years
cardio_disease_1['age'] = cardio_disease_1['age'].astype(float)
cardio_disease_1['age'] = (cardio_disease_1['age']/365).round()
# checking the relation ship between the
```

In [16]:
```
#Distribution of age with the cardio
plt.figure(figsize = (15, 10))
sns.distplot(cardio_disease_1['age'][cardio_disease_1['cardio'] == 0], color = 'blue
sns.distplot(cardio_disease_1['age'][cardio_disease_1['cardio'] == 1], color = 'oran
plt.title("Distribution between age and cardio")
plt.show()
```

```
D:\Anaconda\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplo
t` is a deprecated function and will be removed in a future version. Please adapt yo
ur code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
D:\Anaconda\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplo
t` is a deprecated function and will be removed in a future version. Please adapt yo
ur code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



From the above observation we can see that upon increase in age the chances of getting the cardio vascular disease is high

In [17]:
```
#As per the data gender is given as numerical code  1 : female and 2 : male
# checking the total number of males and females in the given data
cardio_disease_1['gender'].value_counts()
```

Out[17]:
```
1    44286
2    23731
Name: gender, dtype: int64
```

According to the given data the % of most of the persons is female rather than the male and it is obvious that according to this data more affected with the cardio disease is female persons

In [18]:
```
# checking the other categorical variables wihch are given as numerical variables or
print("The number of each categories in cholesterol")
display(cardio_disease_1['cholesterol'].value_counts())
print("The number of each categories in glucose")
display(cardio_disease_1['gluc'].value_counts())
print("The number of each categories in smoke")
display(cardio_disease_1['smoke'].value_counts())
print("The number of each categories in alco")
display(cardio_disease_1['alco'].value_counts())
print("The number of categories in active")
display(cardio_disease_1['active'].value_counts())
```

```
The number of each categories in cholesterol
1    50952
2     9226
3     7839
Name: cholesterol, dtype: int64
The number of each categories in glucose
1    57797
3     5195
2     5025
Name: gluc, dtype: int64
The number of each categories in smoke
0    62052
1     5965
Name: smoke, dtype: int64
The number of each categories in alco
0    64387
1     3630
Name: alco, dtype: int64
The number of categories in active
1    54619
0    13398
Name: active, dtype: int64
```

Usually the cardio vasular disease will be dependent on the above varibales glucose , cholesterol percentage, alchol intake, smoking and physical activity but the data in the dataset is like skewed so according to this data the cardio is not much dependent upon these values the same can be observed in the correlation plot.

In [19]:
```
#dropping the unnecessary columns like id , height, and weight
dropped_cols = ['id', 'height', 'weight']
cardio_disease_1 = cardio_disease_1.drop(dropped_cols, axis = 1)
cardio_disease_1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 68017 entries, 0 to 69999
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   age          68017 non-null  float64
 1   gender       68017 non-null  int64  
 2   ap_hi        68017 non-null  int64  
 3   ap_lo        68017 non-null  int64  
 4   cholesterol  68017 non-null  int64  
 5   gluc         68017 non-null  int64  
 6   smoke        68017 non-null  int64  
 7   alco         68017 non-null  int64
 8   active       68017 non-null  int64
 9   cardio       68017 non-null  int64
 10  bmi          68017 non-null  float64
dtypes: float64(2), int64(9)
memory usage: 6.2 MB
```

In [20]:

```
#correltion between each column
cardio_disease_1.describe()
```

Out[20]:

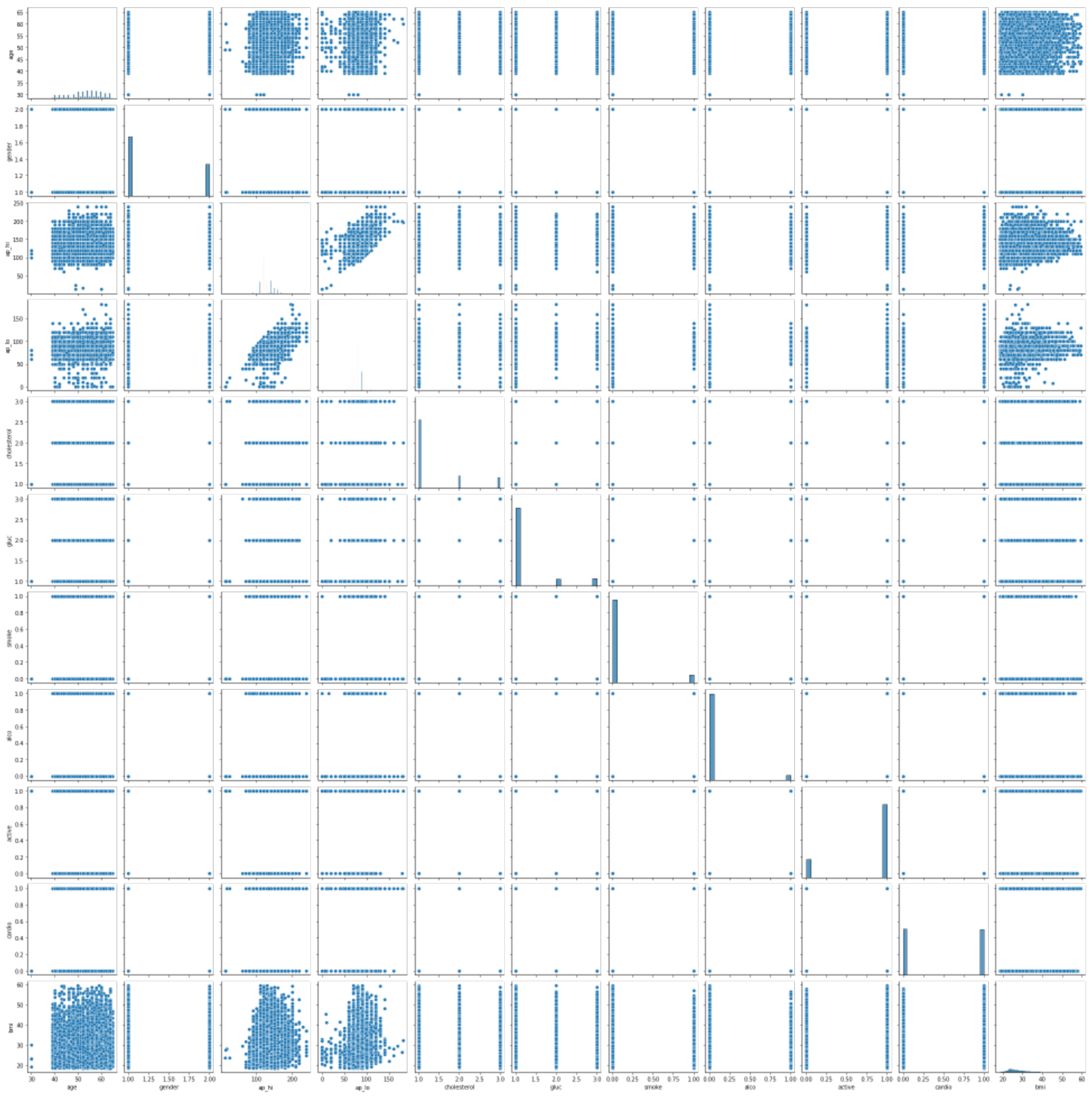
	age	gender	ap_hi	ap_lo	cholesterol	gluc	sr
count	68017.000000	68017.000000	68017.000000	68017.000000	68017.000000	68017.000000	68017.00
mean	53.345811	1.348898	126.768734	81.312951	1.366144	1.226635	0.08
std	6.758224	0.476625	16.683540	9.614005	0.680140	0.572741	0.28
min	30.000000	1.000000	12.000000	0.000000	1.000000	1.000000	0.00
25%	48.000000	1.000000	120.000000	80.000000	1.000000	1.000000	0.00
50%	54.000000	1.000000	120.000000	80.000000	1.000000	1.000000	0.00
75%	58.000000	2.000000	140.000000	90.000000	2.000000	1.000000	0.00
max	65.000000	2.000000	240.000000	182.000000	3.000000	3.000000	1.00

In [21]:

```
sns.pairplot(cardio_disease_1)
```

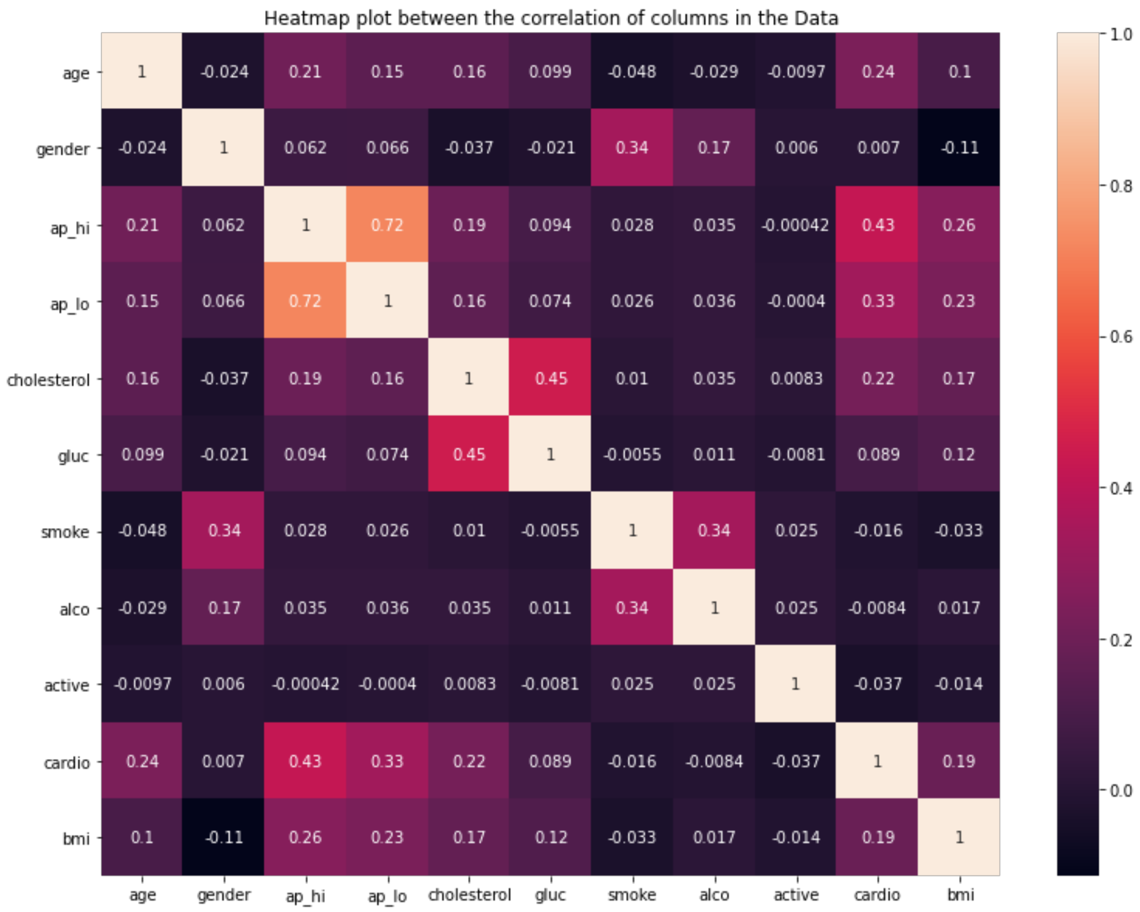
Out[21]:

```
<seaborn.axisgrid.PairGrid at 0x213336c9c10>
```



In [22]:

```
#correlation plot
plt.figure(figsize = (13, 10))
sns.heatmap(cardio_disease_1.corr(), annot = True)
plt.title("Heatmap plot between the correlation of columns in the Data")
plt.show()
```



from the above heatmap we can see that the most concerning feature is ap_hi, ap_lo, age, Bmi

ML- models:

i am using different machine learning algorithms to predict the existence of cardiovascular diseases in patient according to our dataset

1. Decision tree
2. Random forest
3. K - nearest Neighbour
4. SVM

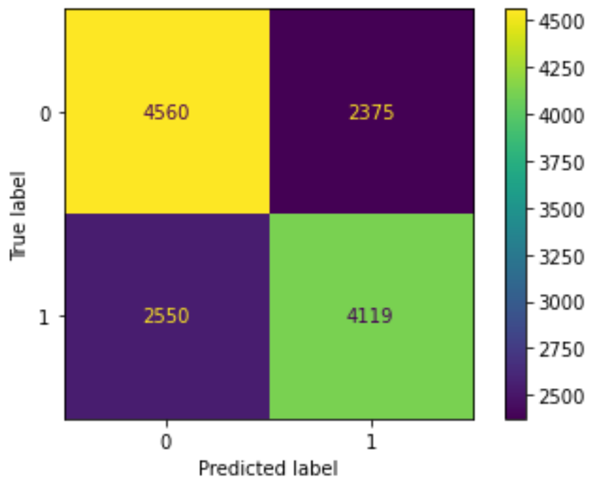
In [61]:
```
# Train test split:
#Seperating the training and test sets
X = cardio_disease_1.drop(['cardio'], axis = 1) # Input variables
y = cardio_disease_1['cardio'] # target variables

# here i am splitting the dataset in the 80:20 ratio
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_st

# the shape of the train and test arrays#
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(54413, 10)
(54413,)
(13604, 10)
(13604,)
```

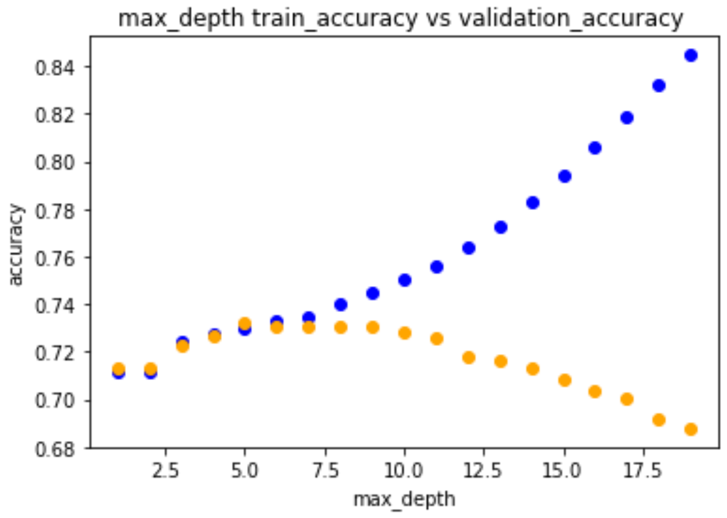
1 Decision Tree

```
In [62]:
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, y_train)
y_pred = decision_tree.predict(X_test)

#metrics calculation:
print(metrics.classification_report(y_test, y_pred)) # classification report

#confusion metrics:
confusion_matrix_dt = metrics.confusion_matrix(y_test, y_pred)
metrics.plot_confusion_matrix(decision_tree, X_test, y_test)

#sensitivity
sensitivity_dt = confusion_matrix_dt[1,1]/(confusion_matrix_dt[1,0] + confusion_matr
print("The sensitivity of the model using Decision tree is:", sensitivity_dt)
accuracy_dt = print("The accuracy of Decision Tree is :", np.round(metrics.accuracy_
```

```
              precision    recall  f1-score   support

           0       0.64      0.66      0.65      6935
           1       0.63      0.62      0.63      6669

    accuracy                           0.64     13604
   macro avg       0.64      0.64      0.64     13604
weighted avg       0.64      0.64      0.64     13604

The sensitivity of the model using Decision tree is: 0.617633828160144
The accuracy of Decision Tree is : 0.64
```



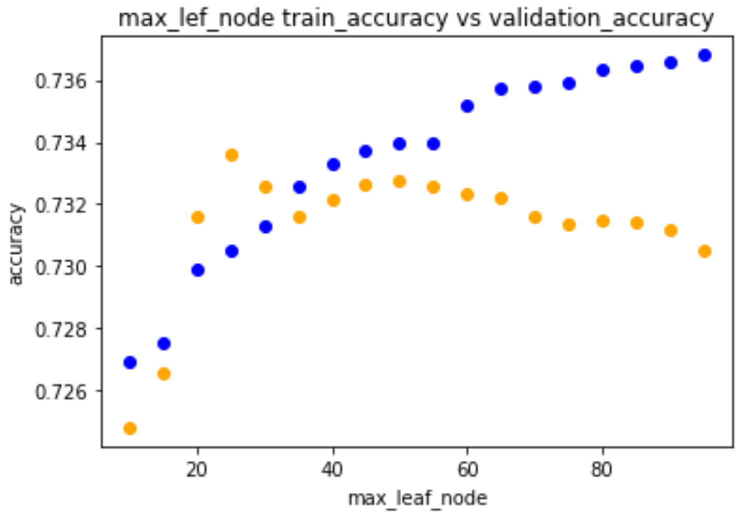
```
In [63]:
#By changing the parameters to the best parameters we can increase the accuracy and
# tuning max_depth hyperparamter
train_accuracy_dt = []
validation_accuracy_dt = []
for i in range(1, 20):
    model = DecisionTreeClassifier(max_depth = i, random_state = 5)
    model.fit(X_train, y_train)
    train_accuracy_dt.append(model.score(X_train, y_train))
    validation_accuracy_dt.append(model.score(X_test, y_test))

#plot between the values #
plt.scatter(range(1, 20), train_accuracy_dt, color = 'blue')
plt.scatter(range(1, 20), validation_accuracy_dt, color = 'orange')
plt.xlabel('max_depth')
plt.ylabel('accuracy')
plt.title("max_depth train_accuracy vs validation_accuracy ")
plt.show()
```



In [64]:

```
# tuning max_leaf_node parameter
train_accuracy_dt1 = []
validation_accuracy_dt1 = []
for j in range(10, 100, 5):
    model = DecisionTreeClassifier(max_leaf_nodes = j , random_state = 5)
    model.fit(X_train, y_train)
    train_accuracy_dt1.append(model.score(X_train, y_train))
    validation_accuracy_dt1.append(model.score(X_test, y_test))
print(train_accuracy_dt1)
print(validation_accuracy_dt1)
#plot between the values #
plt.scatter(range(10,100, 5), train_accuracy_dt1, color = 'blue')
plt.scatter(range(10, 100, 5), validation_accuracy_dt1, color = 'orange')
plt.xlabel('max_leaf_node')
plt.ylabel('accuracy')
plt.title("max_lef_node train_accuracy vs validation_accuracy")
plt.show()
```

```
[0.7269034973260067, 0.7275283480050723, 0.729880727032143, 0.7304871997500597, 0.73
12958300406153, 0.7325822873210446, 0.7333174057670042, 0.733703342951133, 0.7339422
564460699, 0.7339973903295168, 0.7351919578042012, 0.735706540716373, 0.735816808483
2669, 0.7359454542113097, 0.7363313913954386, 0.7364784150846305, 0.736607060812673
5, 0.7368092183853123]
[0.7247868274037048, 0.7265510144075272, 0.7316230520435166, 0.7336077624228168, 0.7
325786533372537, 0.7316230520435166, 0.7321376065862981, 0.7326521611290797, 0.73272
56689209056, 0.7325786533372537, 0.732358129961776, 0.732211114378124, 0.73162305204
35166, 0.7313290208762129, 0.7314760364598647, 0.7314025286680388, 0.73118200529256
1, 0.7305204351661276]
```

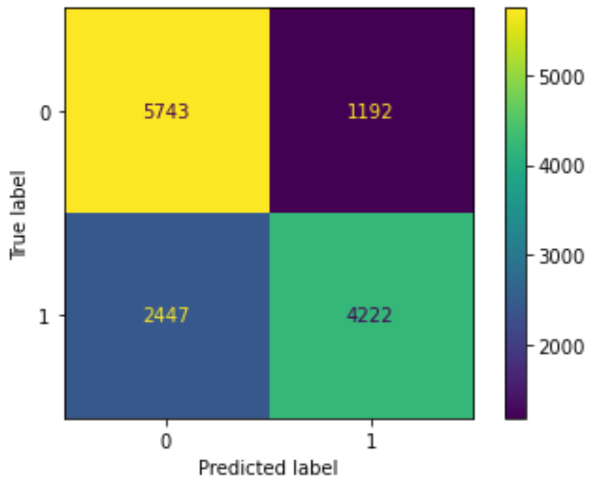


if we check the both graphs for max_leaf_node and max_depth values the max_depth = 5 , max_leaf_node = 25 these points where the train accuracy is incresing and validation accuracy is decreasing so we can use these tuned values in order to get the improved accuracy for the Decision tree model

In [70]:

```
decision_tree_tuned = DecisionTreeClassifier(max_depth = 5, max_leaf_nodes = 25,rand
decision_tree_tuned.fit(X_train, y_train)
y_pred = decision_tree_tuned.predict(X_test)

#metrics calculation:
print(metrics.classification_report(y_test, y_pred)) # classification report

#confusion metrics:
confusion_matrix_dt_tuned = metrics.confusion_matrix(y_test, y_pred)
metrics.plot_confusion_matrix(decision_tree_tuned, X_test, y_test)

#sensitivity
sensitivity_dt_tuned = confusion_matrix_dt_tuned[1,1]/(confusion_matrix_dt_tuned[1,0
print("The sensitivity of the model using Decision tree is:", sensitivity_dt)
accuracy_dt_tuned = metrics.accuracy_score(y_test, y_pred)
print("The accuracy of Decision Tree is :", accuracy_dt_tuned) # accuracy

# by comparing the both models after tuning the accuracy is very much improved
```

```
              precision    recall  f1-score   support

           0       0.70      0.83      0.76      6935
           1       0.78      0.63      0.70      6669

    accuracy                           0.73     13604
   macro avg       0.74      0.73      0.73     13604
weighted avg       0.74      0.73      0.73     13604

The sensitivity of the model using Decision tree is: 0.617633828160144
The accuracy of Decision Tree is : 0.7325051455454278
```



In [28]:

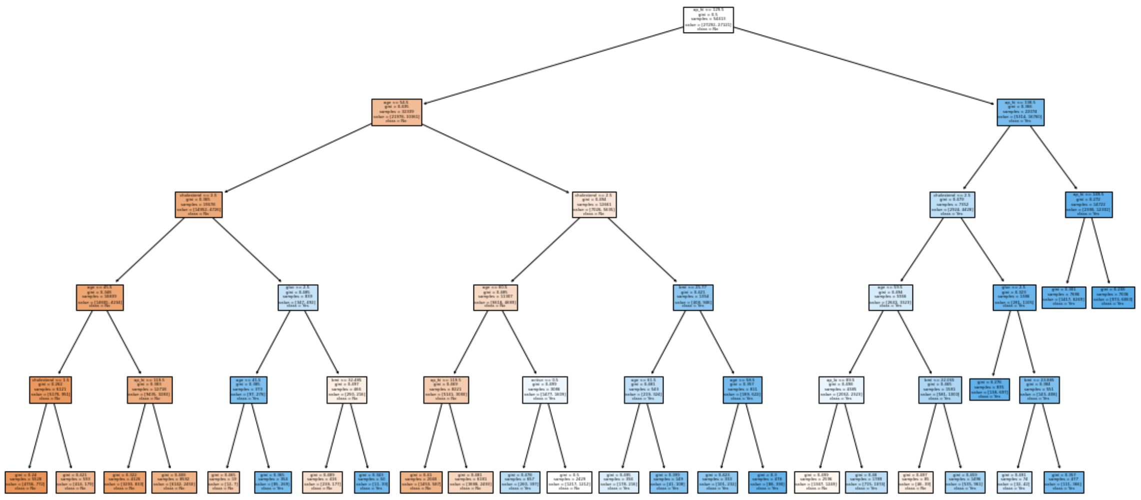
```
#features of importance
x_names = cardio_disease_1.columns.tolist()
x_names.remove('cardio')
y_names = ['No', 'Yes']
feature_importance = {}
for i, name in enumerate(x_names):
    feature_importance.update({name:decision_tree_tuned.feature_importances_[i]})

sorted_order = dict(sorted(feature_importance.items(), key = lambda x:x[1], reverse
print("The important features in the dataset are:")
```

```
for k in sorted_order:
    print(k)
```

```
The important features in the dataset are:
ap_hi
age
cholesterol
bmi
gluc
ap_lo
active
gender
smoke
alco
```

In [29]:

```
from sklearn import tree
figure = plt.figure(figsize = (20,10))
_ = tree.plot_tree(decision_tree_tuned, feature_names = x_names, class_names = y_nam
```

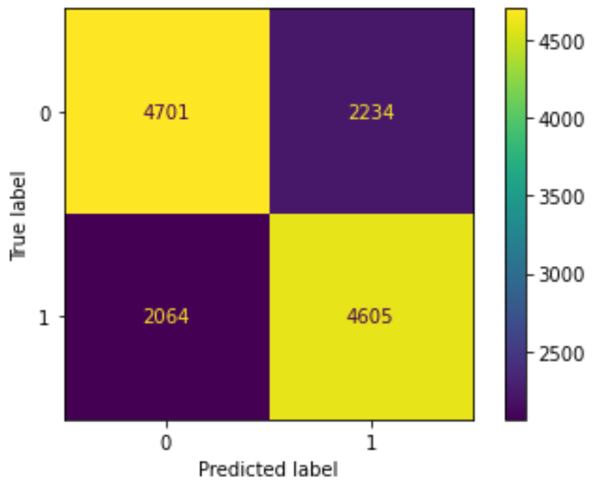


Random forest

In [30]:

```
random_forest = RandomForestClassifier(random_state = 5, n_estimators = 100, bootstr
predict = random_forest.fit(X_train, y_train)
y_pred = predict.predict(X_test)

#metrics:
confusion_metrics_rf = metrics.confusion_matrix(y_test, y_pred)
metrics.plot_confusion_matrix(random_forest, X_test, y_test)

#Sensitivity & Accuracy:
accuracy_rfc = metrics.accuracy_score(y_test, y_pred)
sensitivity_rfc = confusion_metrics_rf[1,1]/(confusion_metrics_rf[1,1]+confusion_met
print("The accuracy of the Random forest classifier is:", accuracy_rfc)
print("The sensitivity of the Random forest classifier is :", sensitivity_rfc)
```

```
The accuracy of the Random forest classifier is: 0.6840635107321376
The sensitivity of the Random forest classifier is : 0.6905083220872694
```



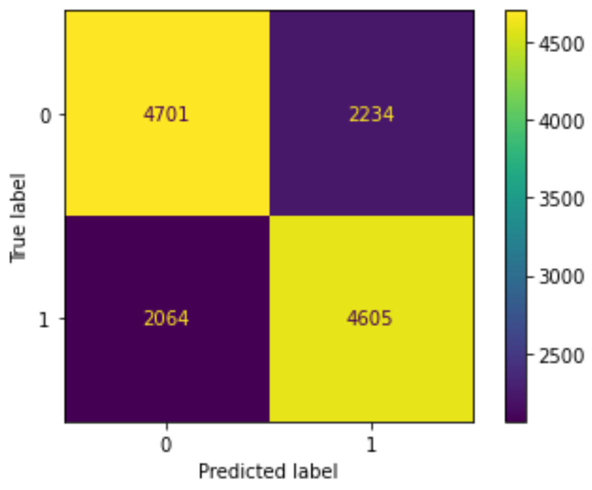
Tuned random forest for best paramters

```
In [35]:
#tuning for best paramters #
from sklearn.model_selection import cross_validate
from sklearn.model_selection import cross_val_score
estimators = {'n_estimators' : [500], 'max_features' :['sqrt'], 'max_depth':[20], 'm
random_forest_classifier_tuned = GridSearchCV(estimator = RandomForestClassifier(),
print(random_forest_classifier_tuned.best_estimator_)
param_selection = random_forest_classifier_tuned.cv_results_['params']
for i in param_selection:
    print(i)
```

```
RandomForestClassifier(max_depth=20, max_features='sqrt', max_leaf_nodes=500,
                       n_estimators=500)
{'max_depth': 20, 'max_features': 'sqrt', 'max_leaf_nodes': 500, 'n_estimators': 50
0}
{'max_depth': 20, 'max_features': 'sqrt', 'max_leaf_nodes': 1000, 'n_estimators': 50
0}
{'max_depth': 20, 'max_features': 'sqrt', 'max_leaf_nodes': 1500, 'n_estimators': 50
0}
```

```
In [37]:
#tuned Random forest :
random_forest_tuned = RandomForestClassifier(random_state = 5, n_estimators = 500, b
                                             max_depth = 20, max_leaf_nodes = 1000,
predict = random_forest_tuned.fit(X_train, y_train)
y_pred = predict.predict(X_test)

#metrics:
confusion_metrics_rf_tuned = metrics.confusion_matrix(y_test, y_pred)
metrics.plot_confusion_matrix(random_forest, X_test, y_test)

#Sensitivity & Accuracy:
accuracy_rfc_tuned = metrics.accuracy_score(y_test, y_pred)
sensitivity_rfc_tuned = confusion_metrics_rf_tuned[1,1]/(confusion_metrics_rf_tuned[
print("The accuracy of the Random forest classifier is:", accuracy_rfc_tuned)
print("The sensitivity of the Random forest classifier is :", sensitivity_rfc_tuned)
```

```
The accuracy of the Random forest classifier is: 0.7340488091737725
The sensitivity of the Random forest classifier is : 0.6905083220872694
```



In [ ]:

```
for i, name in enumerate(x_names):
    feature_importance.update({name:random_forest_tuned.feature_importances_[i]})

sorted_order = dict(sorted(feature_importance.items(), key = lambda x:x[1], reverse
print("The important features in the dataset are:")
for k in sorted_order:
    print(k)
```

Standadized Scalar technique for SVM and KNN models

In [39]:

```
#scaling the train, test dataset for KNN and SVM algorithms
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

k-NearstNeighbor

In [45]:

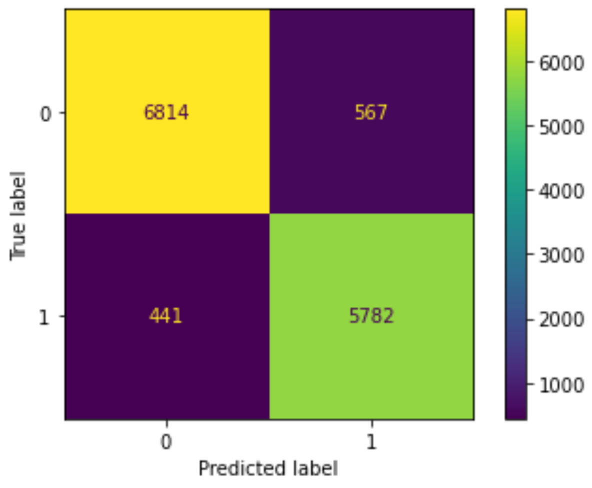
```
#K- nearest Neighbour:
for k in range(1, 50):
    knn = KNeighborsClassifier(n_neighbors = k)
    knn.fit(X_train, y_train)
    score = knn.score(X_test, y_test)
    print(f"KNN accuracy with k: {k}, {score:.2f}")
#upon running randomly with different K values the accuracy is constant after K = 21
```

```
KNN accuracy with k: 1, 0.64
KNN accuracy with k: 2, 0.65
KNN accuracy with k: 3, 0.68
KNN accuracy with k: 4, 0.68
KNN accuracy with k: 5, 0.69
KNN accuracy with k: 6, 0.70
KNN accuracy with k: 7, 0.71
KNN accuracy with k: 8, 0.70
KNN accuracy with k: 9, 0.71
KNN accuracy with k: 10, 0.71
KNN accuracy with k: 11, 0.72
KNN accuracy with k: 12, 0.72
KNN accuracy with k: 13, 0.72
KNN accuracy with k: 14, 0.72
KNN accuracy with k: 15, 0.72
KNN accuracy with k: 16, 0.72
KNN accuracy with k: 17, 0.72
KNN accuracy with k: 18, 0.73
KNN accuracy with k: 19, 0.72
KNN accuracy with k: 20, 0.72
KNN accuracy with k: 21, 0.73
KNN accuracy with k: 22, 0.73
KNN accuracy with k: 23, 0.73
KNN accuracy with k: 24, 0.73
KNN accuracy with k: 25, 0.73
KNN accuracy with k: 26, 0.73
KNN accuracy with k: 27, 0.73
KNN accuracy with k: 28, 0.73
KNN accuracy with k: 29, 0.73
KNN accuracy with k: 30, 0.73
KNN accuracy with k: 31, 0.73
KNN accuracy with k: 32, 0.73
KNN accuracy with k: 33, 0.73
KNN accuracy with k: 34, 0.73
KNN accuracy with k: 35, 0.73
KNN accuracy with k: 36, 0.73
KNN accuracy with k: 37, 0.73
KNN accuracy with k: 38, 0.73
KNN accuracy with k: 39, 0.73
KNN accuracy with k: 40, 0.73
KNN accuracy with k: 41, 0.73
KNN accuracy with k: 42, 0.73
KNN accuracy with k: 43, 0.73
KNN accuracy with k: 44, 0.73
KNN accuracy with k: 45, 0.73
KNN accuracy with k: 46, 0.73
KNN accuracy with k: 47, 0.73
KNN accuracy with k: 48, 0.73
KNN accuracy with k: 49, 0.73
```

In [46]:

```
#Knn with K =21:
knn = KNeighborsClassifier(n_neighbors = 21)
knn.fit(X_train, y_train)
score = knn.score(X_test, y_test)

#confuion matrix
confusion_matrix_knn = metrics.confusion_matrix(y_test, y_pred)
metrics.plot_confusion_matrix(knn, X_test, y_pred)

#accuracy and sensivity
print(f"KNN accuracy with k = 21: {score:.2f}")
sensitivity_knn = confusion_matrix_knn[1,1]/(confusion_matrix_knn[1,1] + confusion_m
print(f"KNN sensitivity with k = 21: {score:.2f}")
```

```
KNN accuracy with k = 21: 0.73
KNN sensitivity with k = 21: 0.73
```



Tuned KNN

In [47]:
```
#tuned KNN
values = []
for i in range(1, 40):
    values.append(i)
from sklearn.model_selection import GridSearchCV
param_grid = {'n_neighbors': values, 'weights': ['uniform', 'distance'], 'metric' :
knn = KNeighborsClassifier()
grid_search = GridSearchCV(knn, param_grid, cv =5)
grid_search.fit(X_train, y_train)
best_params = grid_search.best_params_
print(best_params)
```

```
{'metric': 'manhattan', 'n_neighbors': 35, 'weights': 'uniform'}
```

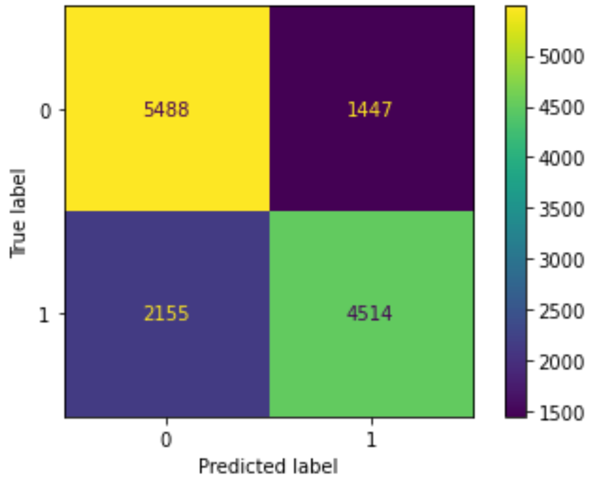
In [50]:
```
#running with the best parameters obtained from grid search CV
knn_tuned = KNeighborsClassifier(**best_params)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)

#confusion matrix
confusion_matrix_knn_tuned = metrics.confusion_matrix(y_test, y_pred)
#metrics.plot_confusion_matrix(knn_tuned, X_test, y_pred)

#Accuracy and sensitivity:
accuracy = metrics.accuracy_score(y_test, y_pred)
print("KNN accuracy with tuned paramteres", accuracy)
sensitivity_knn_tuned = confusion_matrix_knn_tuned[1,1]/(confusion_matrix_knn_tuned[
print(f"KNN sensitivity with tuned parameters:", sensitivity_knn_tuned)
```

```
KNN accuracy with tuned paramteres 0.6931049691267275
KNN sensitivity with tuned parameters: 0.6819613135402609
```

support vector machine

In [55]:
```
#suport vector machine classifier
svm_clf = svm.SVC(kernel = 'rbf')
svm_clf.fit(X_train, y_train)
y_pred = svm_clf.predict(X_test)

#confusion matrix:
confusion_matrix_svm = metrics.confusion_matrix(y_test, y_pred)
metrics.plot_confusion_matrix(svm_clf, X_test, y_test)

#Accuracy:
accuracy = metrics.accuracy_score(y_pred, y_test)
print("the accuracy with svm:",accuracy)
#sensitivity:
sensitivity_svm = confusion_matrix_svm[1,1]/(confusion_matrix_svm[1,0]+ confusion_ma
print("the sensitivty with SVM is", sensitivity_svm)
```

```
the accuracy with svm: 0.7352249338429874
the sensitivty with SVM is 0.6768630979157295
```



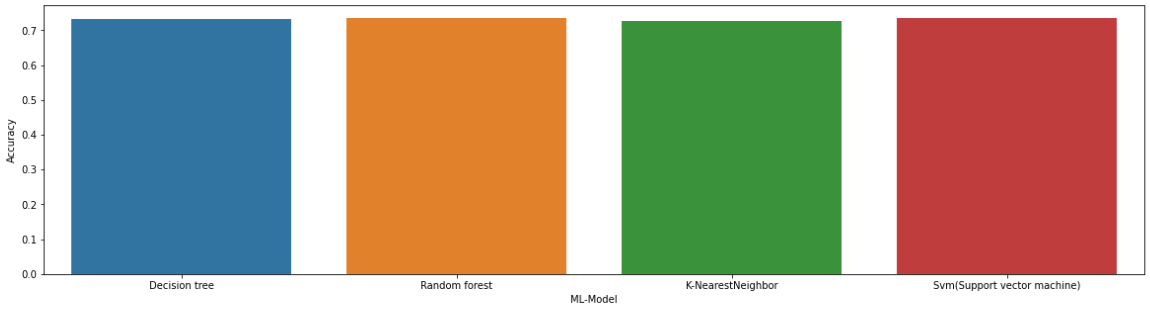
Conclusion

In [71]:
```
# comapring the all 4 machine learning algorithms accuracy
results = pd.DataFrame({'ML-Model' : ['Decision tree', 'Random forest', 'K-NearestNe
                        'Accuracy':[accuracy_dt_tuned,accuracy_rfc_tuned, score, acc
                       'Sensitivity': [sensitivity_dt_tuned, sensitivity_rfc_tuned,
results
```

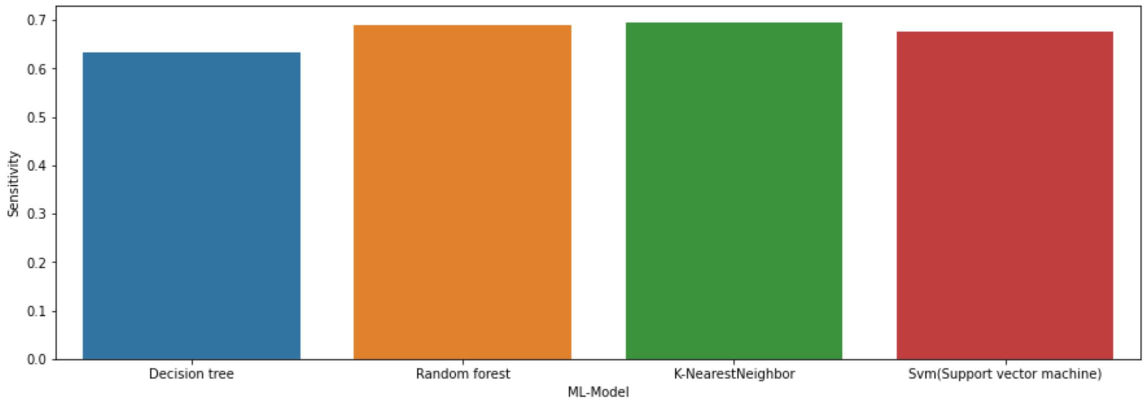
Out[71]:

	ML-Model	Accuracy	Sensitivity
0	Decision tree	0.732505	0.633078
1	Random forest	0.734049	0.690508
2	K-NearestNeighbor	0.726110	0.695307
3	Svm(Support vector machine)	0.735225	0.676863

In [73]:
```
#plotting the results for different models with different accuracies
plt.figure(figsize =(15,5))
sns.barplot(y= results['Accuracy'], x = results['ML-Model'], orient = 'v')
plt.show()
```



In [75]:
```
#plotting the sensitivity of different models#
plt.figure(figsize = (15,5))
sns.barplot(y = results['Sensitivity'], x = results['ML-Model'], orient = "v")
plt.show()
```



By the above we can conclude that all the models aftr tuning are perfomed well and have almost same accuracy. The sensitivity is changing for each but the accuracy is almost same for each algorithm.

Future Refrence

If we can able to train this kind of real-time data we can implement an Application or web browser where we can deploy our model and based on the input values of the people it can predict weather the person is suffering from cardio vascular disease or not.