CardioVascular Disease Detection using Different Machine Learning Algorithms

In [1]:
```
#Importing Libraries
import pandas as pd
from pandas.plotting import scatter_matrix
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

In [2]:
```
#Importing Dataset
df = pd.read_csv('cardio_train.csv')
```

In [3]:
```
df
```

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 [4]:
```
df.shape
```

Out[4]:
```
(70000, 13)
```

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
id             0
age            0
gender         0
height         0
weight         0
ap_hi          0
ap_lo          0
cholesterol    0
gluc           0
smoke          0
alco           0
active         0
cardio         0
dtype: int64
```

In [6]:
```
#Decribing the whole Datset
df.describe()
```

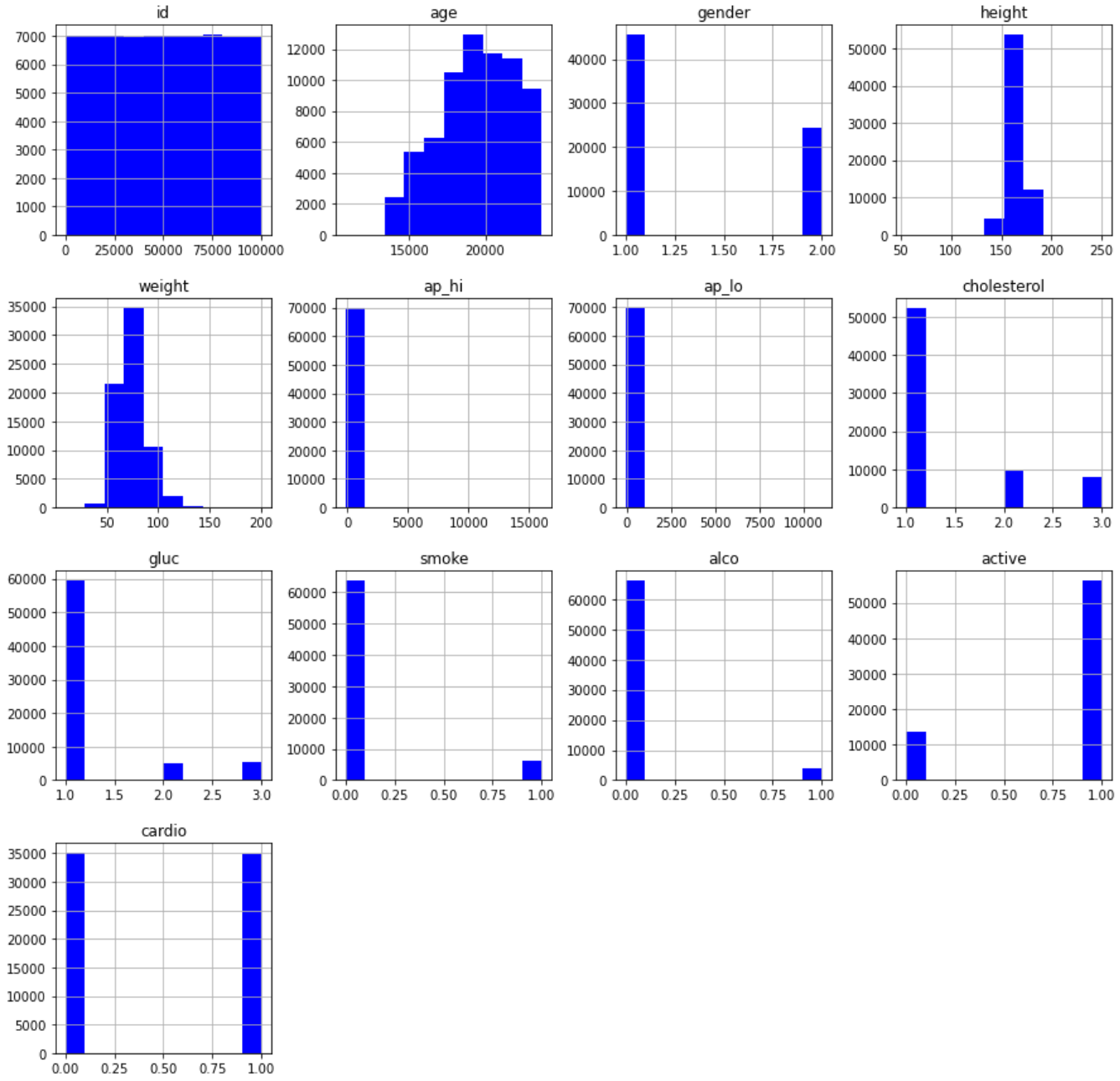
Out[6]:

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
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
mean	49972.419900	19468.865814	1.349571	164.359229	74.205690	128.817286	96.630414	1.366871	1.226457	0.088129	0.053771	0.803729	0.499700
std	28851.302323	2467.251667	0.476838	8.210126	14.395757	154.011419	188.472530	0.680250	0.572270	0.283484	0.225568	0.397179	0.500003
min	0.000000	10798.000000	1.000000	55.000000	10.000000	-150.000000	-70.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000
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
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
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
max	99999.000000	23713.000000	2.000000	250.000000	200.000000	16020.000000	11000.000000	3.000000	3.000000	1.000000	1.000000	1.000000	1.000000

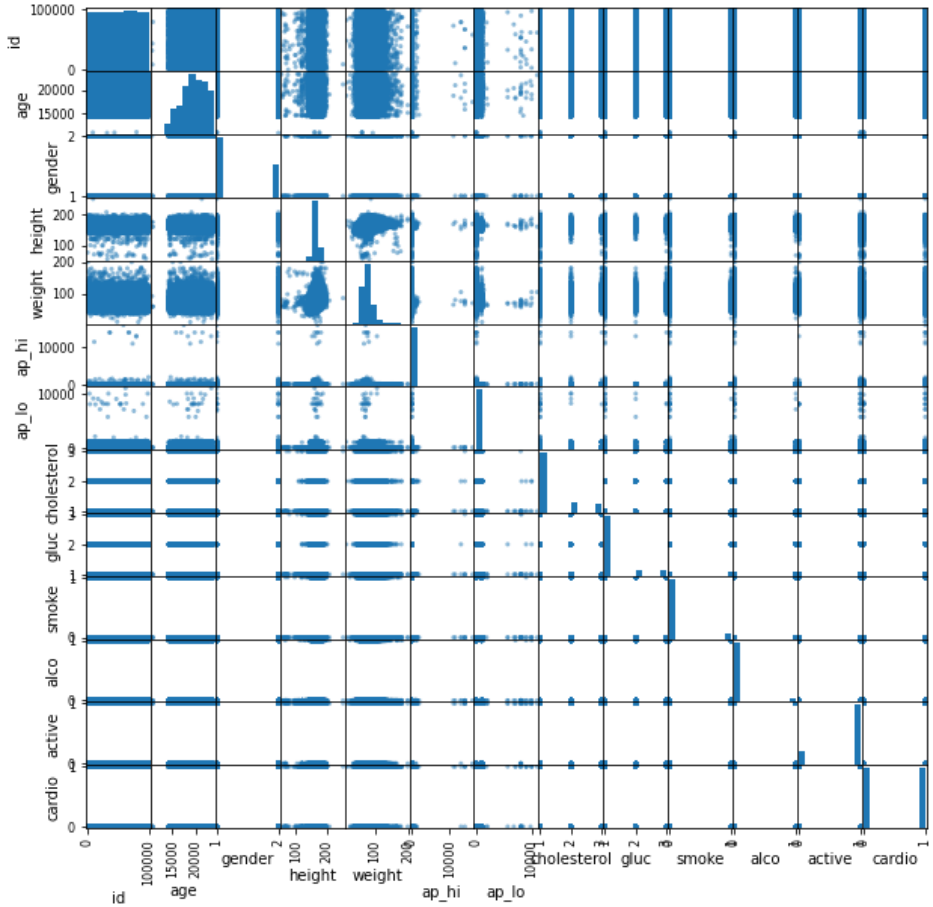
In [7]:
```
#Showing the Number of 0 and 1's, 1(Cardiovascular)
df['cardio'].value_counts()
```

Out[7]:
```
0    35021
1    34979
Name: cardio, dtype: int64
```

In [8]:
```
#Plotting the Dataset
df.hist(figsize = (15, 15),color = 'blue')
plt.show()
```


id
age
gender
height
weight
ap_hi
ap_lo
cholesterol
gluc
smoke
alco
active
cardio

In [9]:
```
#Scattering the Plots
scatter_matrix(df, figsize = (10, 10))
plt.show()
```



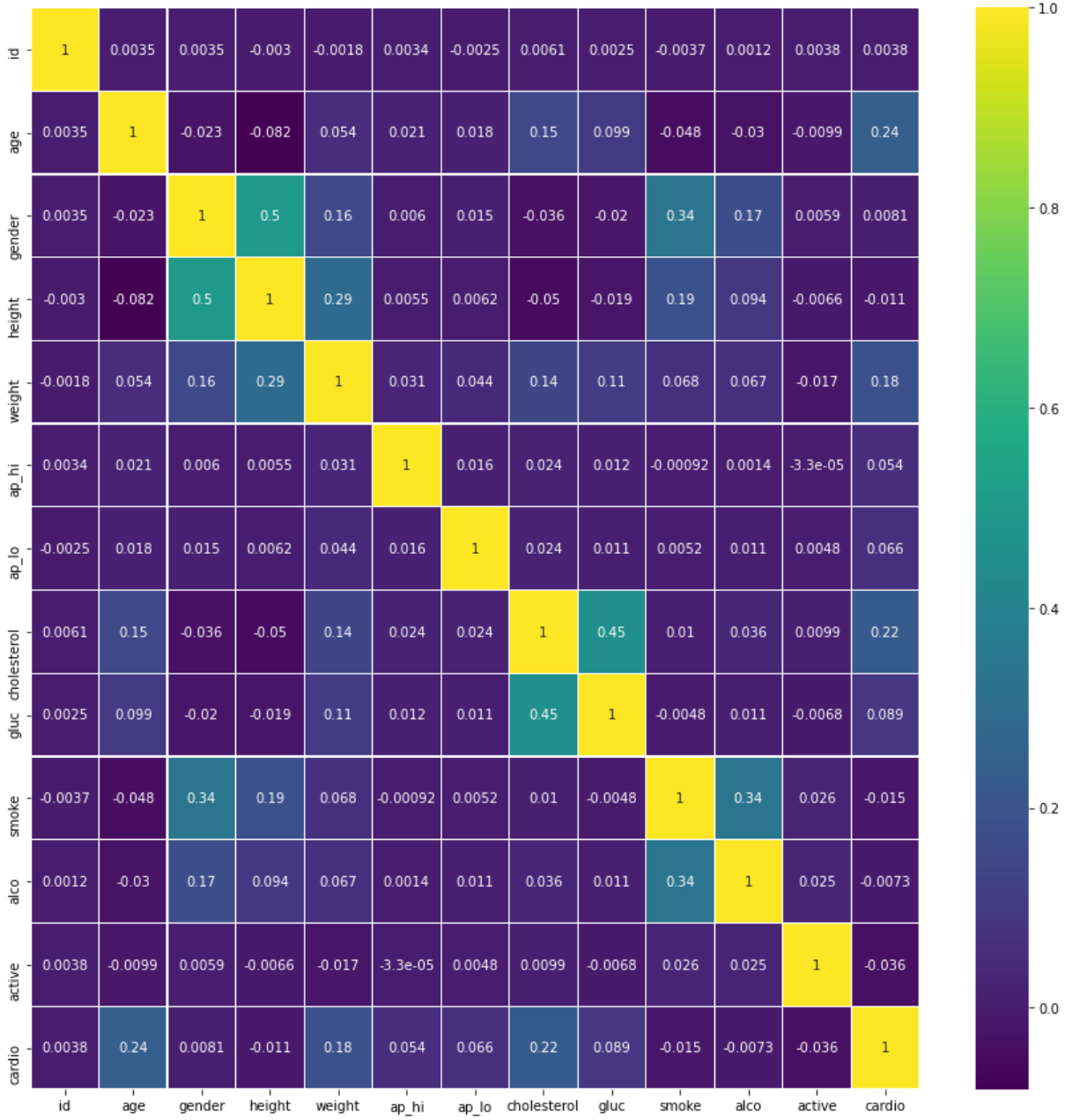
In [10]:
```
#Correlation
df.corr()
```

Out[10]:

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
id	1.000000	0.003457	0.003502	-0.003038	-0.001830	0.003356	-0.002529	0.006106	0.002467	-0.003699	0.001210	0.003755	0.003799
age	0.003457	1.000000	-0.022811	-0.081515	0.053684	0.020764	0.017647	0.154424	0.098703	-0.047633	-0.029723	-0.009927	0.238159
gender	0.003502	-0.022811	1.000000	0.499033	0.155406	0.006005	0.015254	-0.035821	-0.020491	0.338135	0.170966	0.005866	0.008109
height	-0.003038	-0.081515	0.499033	1.000000	0.290968	0.005488	0.006150	-0.050226	-0.018595	0.187989	0.094419	-0.006570	-0.010821
weight	-0.001830	0.053684	0.155406	0.290968	1.000000	0.030702	0.043710	0.141768	0.106857	0.067780	0.067113	-0.016867	0.181660
ap_hi	0.003356	0.020764	0.006005	0.005488	0.030702	1.000000	0.016086	0.023778	0.011841	-0.000922	0.001408	-0.000033	0.054475
ap_lo	-0.002529	0.017647	0.015254	0.006150	0.043710	0.016086	1.000000	0.024019	0.010806	0.005186	0.010601	0.004780	0.065719
cholesterol	0.006106	0.154424	-0.035821	-0.050226	0.141768	0.023778	0.024019	1.000000	0.451578	0.010354	0.035760	0.009911	0.221147
gluc	0.002467	0.098703	-0.020491	-0.018595	0.106857	0.011841	0.010806	0.451578	1.000000	-0.004756	0.011246	-0.006770	0.089307
smoke	-0.003699	-0.047633	0.338135	0.187989	0.067780	-0.000922	0.005186	0.010354	-0.004756	1.000000	0.340094	0.025858	-0.015486
alco	0.001210	-0.029723	0.170966	0.094419	0.067113	0.001408	0.010601	0.035760	0.011246	0.340094	1.000000	0.025476	-0.007330
active	0.003755	-0.009927	0.005866	-0.006570	-0.016867	-0.000033	0.004780	0.009911	-0.006770	0.025858	0.025476	1.000000	-0.035653
cardio	0.003799	0.238159	0.008109	-0.010821	0.181660	0.054475	0.065719	0.221147	0.089307	-0.015486	-0.007330	-0.035653	1.000000

In [11]:
```
#Correlation Matrix Visualization
corrmat = df.corr()
plt.figure(figsize = (15, 15))
sns.heatmap(corrmat, cmap = 'viridis', annot = True, linewidths = '.25')
```

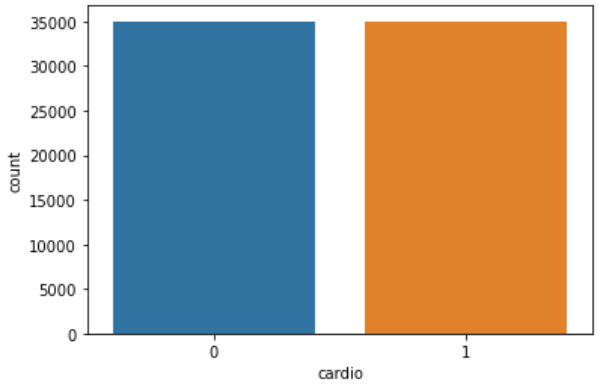
Out[11]: <AxesSubplot:>


id 1 0.0035 0.0035 -0.003 -0.0018 0.0034 -0.0025 0.0061 0.0025 -0.0037 0.0012 0.0038 0.0038
age 0.0035 1 -0.023 -0.082 0.054 0.021 0.018 0.15 0.099 -0.048 -0.03 -0.0099 0.24
gender 0.0035 -0.023 1 0.5 0.16 0.006 0.015 -0.036 -0.02 0.34 0.17 0.0059 0.0081
height -0.003 -0.082 0.5 1 0.29 0.0055 0.0062 -0.05 -0.019 0.19 0.094 -0.0066 -0.011
weight -0.0018 0.054 0.16 0.29 1 0.031 0.044 0.14 0.11 0.068 0.067 -0.017 0.18
ap_hi 0.0034 0.021 0.006 0.0055 0.031 1 0.016 0.024 0.012 -0.00092 0.0014 -3.3e-05 0.054
ap_lo -0.0025 0.018 0.015 0.0062 0.044 0.016 1 0.024 0.011 0.0052 0.011 0.0048 0.066
cholesterol 0.0061 0.15 -0.036 -0.05 0.14 0.024 0.024 1 0.45 0.01 0.036 0.0099 0.22
gluc 0.0025 0.099 -0.02 -0.019 0.11 0.012 0.011 0.45 1 -0.0048 0.011 -0.0068 0.089
smoke -0.0037 -0.048 0.34 0.19 0.068 -0.00092 0.0052 0.01 -0.0048 1 0.34 0.026 -0.015
alco 0.0012 -0.03 0.17 0.094 0.067 0.0014 0.011 0.036 0.011 0.34 1 0.025 -0.0073
active 0.0038 -0.0099 0.0059 -0.0066 -0.017 -3.3e-05 0.0048 0.0099 -0.0068 0.026 0.025 1 -0.036
cardio 0.0038 0.24 0.0081 -0.011 0.18 0.054 0.066 0.22 0.089 -0.015 -0.0073 -0.036 1
id age gender height weight ap_hi ap_lo cholesterol gluc smoke alco active cardio
1.0 0.8 0.6 0.4 0.2 0.0

In [12]:
```
sns.countplot(df['cardio'])
```

```
C:\Users\sahithya\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

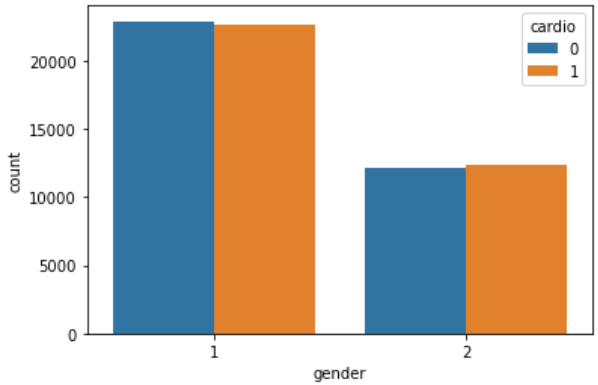
Out[12]: <AxesSubplot:xlabel='cardio', ylabel='count'>



Visualizing each Columns with the Output Column

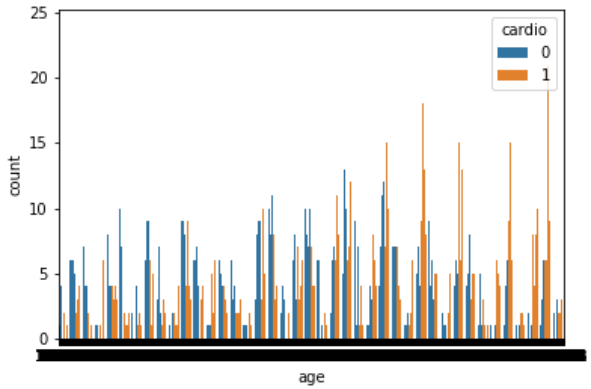
In [13]:
```
sns.countplot(data=df, x="gender", hue="cardio")
```

Out[13]: <AxesSubplot:xlabel='gender', ylabel='count'>



In [14]:
```
sns.countplot(data = df, x = 'age', hue = 'cardio')
```

Out[14]:
```
<AxesSubplot:xlabel='age', ylabel='count'>
```



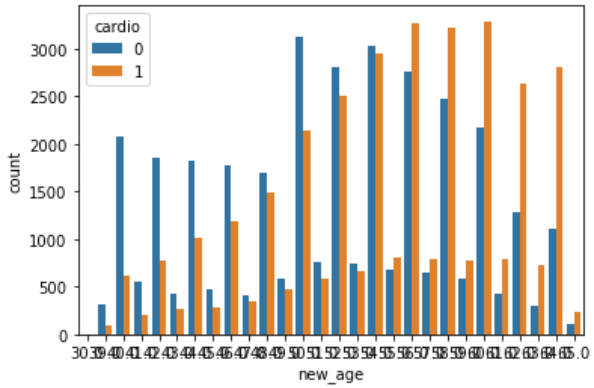
In [15]:
```
#Converting the age into round figure
df['new_age'] = (df['age']/365).round(0)
```

In [16]:
```
df['new_age']
```

Out[16]:
```
0        50.0
1        55.0
2        52.0
3        48.0
4        48.0
         ...
69995    53.0
69996    62.0
69997    52.0
69998    61.0
69999    56.0
Name: new_age, Length: 70000, dtype: float64
```

In [17]:
```
sns.countplot(data = df, x = df['new_age'], hue = 'cardio')
```

Out[17]: <AxesSubplot:xlabel='new_age', ylabel='count'>



In [18]:
```
df.drop(['new_age'], axis = 'columns')
```

Out[18]:

	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

Dividing Features and Label Comuns

In [19]:
```
x = df.drop(['cardio', 'id'], axis = 'columns')
```

In [20]:
```
#Feature Columns
x
```

Out[20]:

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	new_age
0	18393	2	168	62.0	110	80	1	1	0	0	1	50.0
1	20228	1	156	85.0	140	90	3	1	0	0	1	55.0
2	18857	1	165	64.0	130	70	3	1	0	0	0	52.0
3	17623	2	169	82.0	150	100	1	1	0	0	1	48.0
4	17474	1	156	56.0	100	60	1	1	0	0	0	48.0
...	...	...	...	...	...	...	...	...	...	...	...	...
69995	19240	2	168	76.0	120	80	1	1	1	0	1	53.0
69996	22601	1	158	126.0	140	90	2	2	0	0	1	62.0
69997	19066	2	183	105.0	180	90	3	1	0	1	0	52.0
69998	22431	1	163	72.0	135	80	1	2	0	0	0	61.0
69999	20540	1	170	72.0	120	80	2	1	0	0	1	56.0

70000 rows × 12 columns

In [21]:
```
y = df['cardio']
```

In [22]:
```
#Output Column
y
```

Out[22]:
```
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
```

Dividing into Training and Testing Data

In [23]:
```
from sklearn.model_selection import train_test_split
```

In [24]:
```
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size = .30, random_state = 1)
```

In [25]:
```
#Showing xtrain
xtrain
```

Out[25]:

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	new_age
23561	16136	2	169	71.0	100	80	1	1	1	0	1	44.0
34858	14615	1	158	69.0	140	80	2	1	0	0	1	40.0
54953	20507	1	164	65.0	120	80	1	1	0	0	1	56.0
59230	16720	1	153	53.0	100	60	1	1	0	0	1	46.0
1730	21050	1	159	71.0	140	90	1	1	0	0	1	58.0
...	...	...	...	...	...	...	...	...	...	...	...	...
49100	21289	2	175	78.0	120	80	1	1	0	0	1	58.0
20609	19116	1	164	68.0	120	80	1	1	0	0	0	52.0
21440	18049	2	178	82.0	120	80	1	1	0	0	1	49.0
50057	21957	1	169	77.0	120	80	1	1	0	0	0	60.0
5192	20671	1	174	65.0	160	90	2	2	0	0	1	57.0

49000 rows × 12 columns

In [26]:
```
#Showing ytrain
ytrain
```

Out[26]:
```
23561    0
34858    1
54953    0
59230    0
1730     1
        ..
49100    1
20609    0
21440    0
50057    1
5192     1
Name: cardio, Length: 49000, dtype: int64
```

Model Developing using Random Forest

In [27]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [28]:
```
rfc = RandomForestClassifier(n_estimators = 100)
```

In [29]:
```
rfc.fit(xtrain, ytrain)
```

Out[29]:
```
RandomForestClassifier()
```

In [30]:
```
pred = rfc.predict(xtest)
```

In [31]:
```
rfc.score(xtest, ytest)
```

Out[31]:
```
0.7175238095238096
```

In [32]:
```
cr = classification_report(ytest, pred)
```

In [33]:
```
print (cr)
```

```
              precision    recall  f1-score   support

           0       0.70      0.74      0.72     10352
           1       0.73      0.69      0.71     10648

    accuracy                           0.72     21000
   macro avg       0.72      0.72      0.72     21000
weighted avg       0.72      0.72      0.72     21000
```

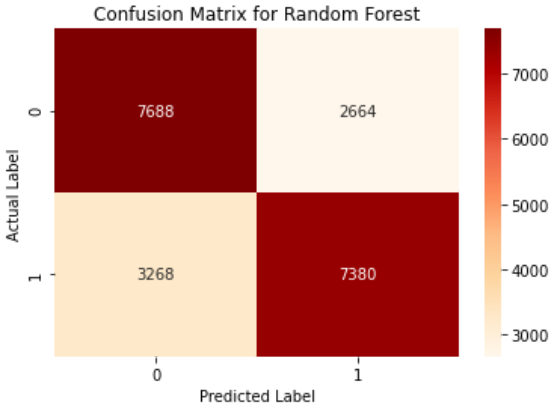
In [34]:
```
cm = confusion_matrix(ytest, pred)
```

In [35]:
```
print (cm)
```

```
[[7688 2664]
 [3268 7380]]
```

In [36]:
```
#Visualization of the Confusion Matrix
p = sns.heatmap(pd.DataFrame(cm), annot = True, cmap = 'OrRd', fmt = 'g')
plt.title('Confusion Matrix for Random Forest')
plt.xlabel('Predicted Label')
plt.ylabel('Actual Label')
```

Out[36]:
```
Text(33.0, 0.5, 'Actual Label')
```



Model Developing using Decision Tree

In [37]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [38]:
```
dtc = DecisionTreeClassifier()
```

In [39]:
```
dtc.fit(xtrain, ytrain)
```

Out[39]:
```
DecisionTreeClassifier()
```

In [40]:
```
pred = dtc.predict(xtest)
```

In [41]:
```
dtc.score(xtest, ytest)
```

Out[41]:
```
0.6346666666666667
```

In [42]:
```
cr = classification_report(ytest, pred)
```

In [43]:
```
print (cr)
```

```
              precision    recall  f1-score   support

           0       0.63      0.64      0.63     10352
           1       0.64      0.63      0.64     10648

    accuracy                           0.63     21000
   macro avg       0.63      0.63      0.63     21000
weighted avg       0.63      0.63      0.63     21000
```

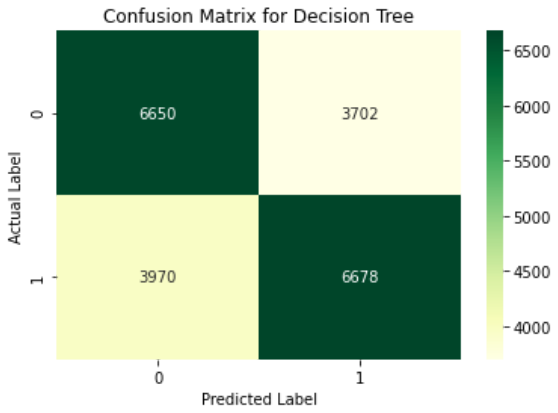
In [44]:
```
cm = confusion_matrix(ytest, pred)
```

In [45]:
```
print (cm)
```

```
[[6650 3702]
 [3970 6678]]
```

In [46]:
```
#Visualization of the Confusion Matrix
p = sns.heatmap(pd.DataFrame(cm), annot = True, cmap = 'YlGn', fmt = 'g')
plt.title('Confusion Matrix for Decision Tree')
plt.xlabel('Predicted Label')
plt.ylabel('Actual Label')
```

Out[46]: Text(33.0, 0.5, 'Actual Label')



Model Developing using Support Vector Machine

In [47]:
```
from sklearn.svm import SVC
```

In [48]:
```
svm = SVC()
```

In [49]:
```
svm.fit(xtrain, ytrain)
```

Out[49]:
```
SVC()
```

In [50]:
```
svm.score(xtest, ytest)
```

Out[50]:
```
0.6046666666666667
```

In [51]:
```
pred = svm.predict(xtest)
```

In [52]:
```
cr = classification_report(ytest, pred)
```

In [53]:
```
print (cr)
```

```
              precision    recall  f1-score   support

           0       0.59      0.66      0.62     10352
           1       0.63      0.55      0.58     10648

    accuracy                           0.60     21000
   macro avg       0.61      0.61      0.60     21000
weighted avg       0.61      0.60      0.60     21000
```

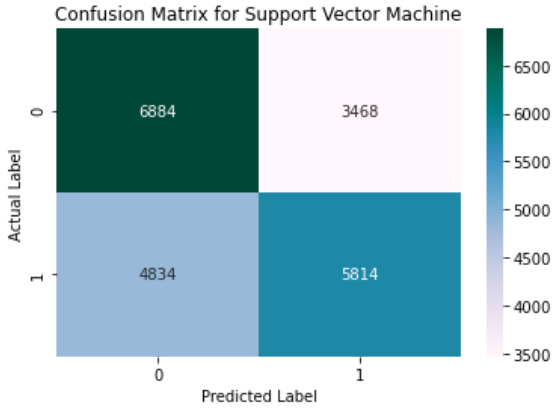
In [54]:
```
cm = confusion_matrix(ytest, pred)
```

In [55]:
```
print (cm)
```

```
[[6884 3468]
 [4834 5814]]
```

In [56]:
```
#Visualization of the Confusion Matrix
p = sns.heatmap(pd.DataFrame(cm), annot = True, cmap = 'PuBuGn', fmt = 'g')
plt.title('Confusion Matrix for Support Vector Machine')
plt.xlabel('Predicted Label')
plt.ylabel('Actual Label')
```

Out[56]:
```
Text(33.0, 0.5, 'Actual Label')
```



Model Developing using Logistic Regression

In [57]:
```
from sklearn.linear_model import LogisticRegression
```

In [58]:
```
lr = LogisticRegression()
```

In [59]:
```
lr.fit(xtrain, ytrain)
```

```
C:\Users\sahithya\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[59]:
```
LogisticRegression()
```

In [60]:
```
lr.score(xtest, ytest)
```

Out[60]:
```
0.7036190476190476
```

In [61]:
```
pred = lr.predict(xtest)
```

In [62]:
```
cr = classification_report(ytest, pred)
```

In [63]:
```
print (cr)
```

```
              precision    recall  f1-score   support

           0       0.68      0.75      0.71     10352
           1       0.73      0.66      0.69     10648

    accuracy                           0.70     21000
   macro avg       0.71      0.70      0.70     21000
weighted avg       0.71      0.70      0.70     21000
```

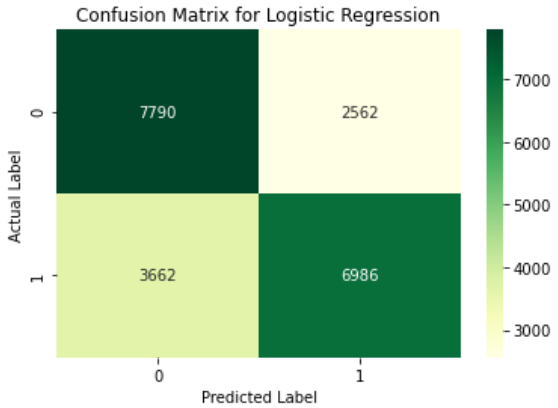
In [64]:
```
cm = confusion_matrix(ytest, pred)
```

In [65]:
```
print (cm)
```

```
[[7790 2562]
 [3662 6986]]
```

In [66]:
```
#Visualization of the Confusion Matrix
p = sns.heatmap(pd.DataFrame(cm), annot = True, cmap = 'YlGn', fmt = 'g')
plt.title('Confusion Matrix for Logistic Regression')
plt.xlabel('Predicted Label')
plt.ylabel('Actual Label')
```

Out[66]: Text(33.0, 0.5, 'Actual Label')



Model Developing using Gaussian Naive Bayes

In [67]:
```
from sklearn.naive_bayes import GaussianNB
```

In [68]:
```
gnb = GaussianNB()
```

In [69]:
```
gnb.fit(xtrain, ytrain)
```

Out[69]:
```
GaussianNB()
```

In [70]:
```
gnb.score(xtest, ytest)
```

Out[70]:
```
0.5910952380952381
```

In [71]:
```
pred = gnb.predict(xtest)
```

In [72]:
```
cr = classification_report(ytest, pred)
```

In [73]:
```
print (cr)
```

```
              precision    recall  f1-score   support

           0       0.55      0.89      0.68     10352
           1       0.74      0.30      0.43     10648

    accuracy                           0.59     21000
   macro avg       0.64      0.60      0.55     21000
weighted avg       0.65      0.59      0.55     21000
```

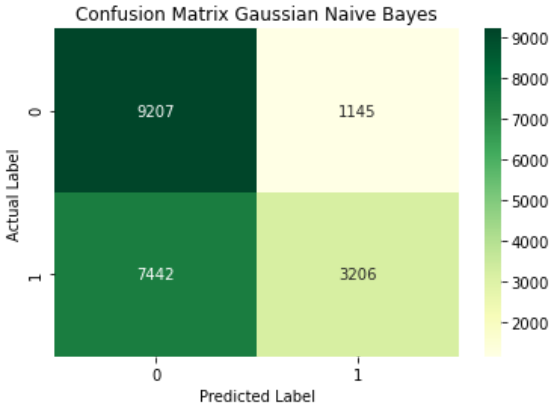
In [74]:
```
cm = confusion_matrix(ytest, pred)
```

In [75]:
```
print (cm)
```

```
[[9207 1145]
 [7442 3206]]
```

In [76]:
```
#Visualization of the Confusion Matrix
p = sns.heatmap(pd.DataFrame(cm), annot = True, cmap = 'YlGn', fmt = 'g')
plt.title('Confusion Matrix Gaussian Naive Bayes')
plt.xlabel('Predicted Label')
plt.ylabel('Actual Label')
```

Out[76]:
```
Text(33.0, 0.5, 'Actual Label')
```



Model Developing using K-Nearest Neighbors

In [77]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [78]:
```
knn = KNeighborsClassifier()
```

In [79]:
```
knn.fit(xtrain, ytrain)
```

Out[79]:
```
KNeighborsClassifier()
```

In [80]:
```
knn.score(xtest, ytest)
```

Out[80]:
```
0.682047619047619
```

In [81]:
```
pred = knn.predict(xtest)
```

In [82]:
```
cr = classification_report(ytest, pred)
```

In [83]:
```
print (cr)
```

```
              precision    recall  f1-score   support

           0       0.67      0.71      0.69     10352
           1       0.70      0.66      0.68     10648

    accuracy                           0.68     21000
   macro avg       0.68      0.68      0.68     21000
weighted avg       0.68      0.68      0.68     21000
```

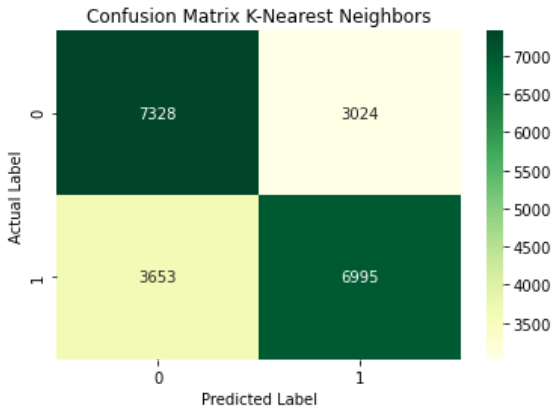
In [84]:
```
cm = confusion_matrix(ytest, pred)
```

In [85]:
```
print (cm)
```

```
[[7328 3024]
 [3653 6995]]
```

In [86]:
```
#Visualization of the Confusion Matrix
p = sns.heatmap(pd.DataFrame(cm), annot = True, cmap = 'YlGn', fmt = 'g')
plt.title('Confusion Matrix K-Nearest Neighbors')
plt.xlabel('Predicted Label')
plt.ylabel('Actual Label')
```

Out[86]: Text(33.0, 0.5, 'Actual Label')



Model Developing using Linear Discriminant Analysis

In [87]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

In [88]:
```
lda = LinearDiscriminantAnalysis()
```

In [89]:
```
lda.fit(xtrain, ytrain)
```

Out[89]:
```
LinearDiscriminantAnalysis()
```

In [90]:
```
lda.score(xtest, ytest)
```

Out[90]:
```
0.6458095238095238
```

In [91]:
```
pred = lda.predict(xtest)
```

In [92]:
```
cr = classification_report(ytest, pred)
```

In [93]:
```
print (cr)
```

```
              precision    recall  f1-score   support

           0       0.63      0.69      0.66     10352
           1       0.67      0.60      0.63     10648

    accuracy                           0.65     21000
   macro avg       0.65      0.65      0.65     21000
weighted avg       0.65      0.65      0.65     21000
```

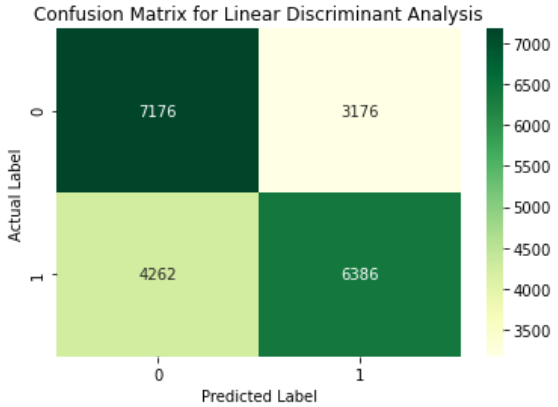
In [94]:
```
cm = confusion_matrix(ytest, pred)
```

In [95]:
```
print (cm)
```

```
[[7176 3176]
 [4262 6386]]
```

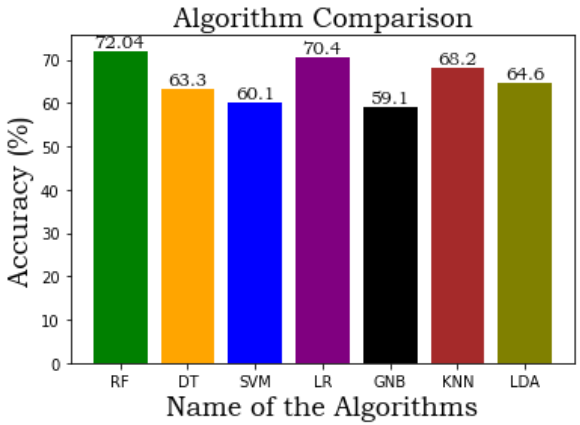
In [96]:
```
#Visualization of the Confusion Matrix
p = sns.heatmap(pd.DataFrame(cm), annot = True, cmap = 'YlGn', fmt = 'g')
plt.title('Confusion Matrix for Linear Discriminant Analysis')
plt.xlabel('Predicted Label')
plt.ylabel('Actual Label')
```

Out[96]:
```
Text(33.0, 0.5, 'Actual Label')
```



In [97]:
```
x = np.array(["RF", "DT", "SVM", "LR", "GNB", "KNN", "LDA"])
y = np.array([72.04, 63.3, 60.1, 70.4, 59.1, 68.2, 64.6])
colors_list = ['Green','Orange', 'Blue', 'Purple', 'Black', 'Brown', 'Olive']
plt.xlabel('Name of the Algorithms', fontname="Bookman Old Style", fontsize=18)
plt.title('Algorithm Comparison', fontname="Bookman Old Style", fontsize=18)
plt.ylabel('Accuracy (%)', fontname="Bookman Old Style", fontsize=18)
pb = plt.bar(x, y, color = colors_list)
for i in range(len(x)):
    plt.text(i, y[i], y[i], ha = "center", va = "bottom", fontname="Bookman Old Style", fontsize = 12)

plt.show()
```



In [ ]: