In [3]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_c
sv)
import seaborn as sns
from matplotlib import pyplot as plt
import os
```

First we import the dataset.

In [6]:
```
dataset = pd.read_csv('/Users/kad99kev/Desktop/Cardio ML/cardio_tra
in.csv', sep = ';')
```

Now checking the various features of the dataset.

In [7]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70000 entries, 0 to 69999
Data columns (total 13 columns):
id             70000 non-null int64
age            70000 non-null int64
gender         70000 non-null int64
height         70000 non-null int64
weight         70000 non-null float64
ap_hi          70000 non-null int64
ap_lo          70000 non-null int64
cholesterol    70000 non-null int64
gluc           70000 non-null int64
smoke          70000 non-null int64
alco           70000 non-null int64
active         70000 non-null int64
cardio         70000 non-null int64
dtypes: float64(1), int64(12)
memory usage: 6.9 MB
```

- RangeIndex: 70000 entries, 0 to 69999
- Data columns (total 13 columns):
- id 70000 non-null int64
- age 70000 non-null int64
- gender 70000 non-null int64
- height 70000 non-null int64
- weight 70000 non-null float64
- ap_hi 70000 non-null int64
- ap_lo 70000 non-null int64
- cholesterol 70000 non-null int64
- gluc 70000 non-null int64
- smoke 70000 non-null int64
- alco 70000 non-null int64
- active 70000 non-null int64
- cardio 70000 non-null int64
- dtypes: float64(1), int64(12)

In [8]:
```
dataset.describe()
```

Out[8]:

	id	age	gender	height	weight	ap_h
count	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.00000
mean	49972.419900	19468.865814	1.349571	164.359229	74.205690	128.81728
std	28851.302323	2467.251667	0.476838	8.210126	14.395757	154.01141
min	0.000000	10798.000000	1.000000	55.000000	10.000000	-150.00000
25%	25006.750000	17664.000000	1.000000	159.000000	65.000000	120.00000
50%	50001.500000	19703.000000	1.000000	165.000000	72.000000	120.00000
75%	74889.250000	21327.000000	2.000000	170.000000	82.000000	140.00000
max	99999.000000	23713.000000	2.000000	250.000000	200.000000	16020.00000

In [9]:
```
dataset.head()
```

Out[9]:

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

As you can see:

- The age is given by days.
- Gender is denoted by 1 and 2 (Need to find what each number stands for).
- Height is in centimeters, as integer values.
- Weight is in kilograms, as float values.
- Systolic (ap_hi) and Diastolic (ap_lo) blood pressure, as integer values.
- Cholesterol and Glucose levels indicated by zone , as integer values.
- Smoking, Alcholic intake and Physical Activity as Binary values.
- Presence or absence of cardiovascular disease as Binary values.

Now let's check the affect each parameter has on the CVD.

First starting with age, by converting days to years.

In [10]:
```
dataset['years'] = (dataset['age']/360).round().astype(int)
dataset.head()
```

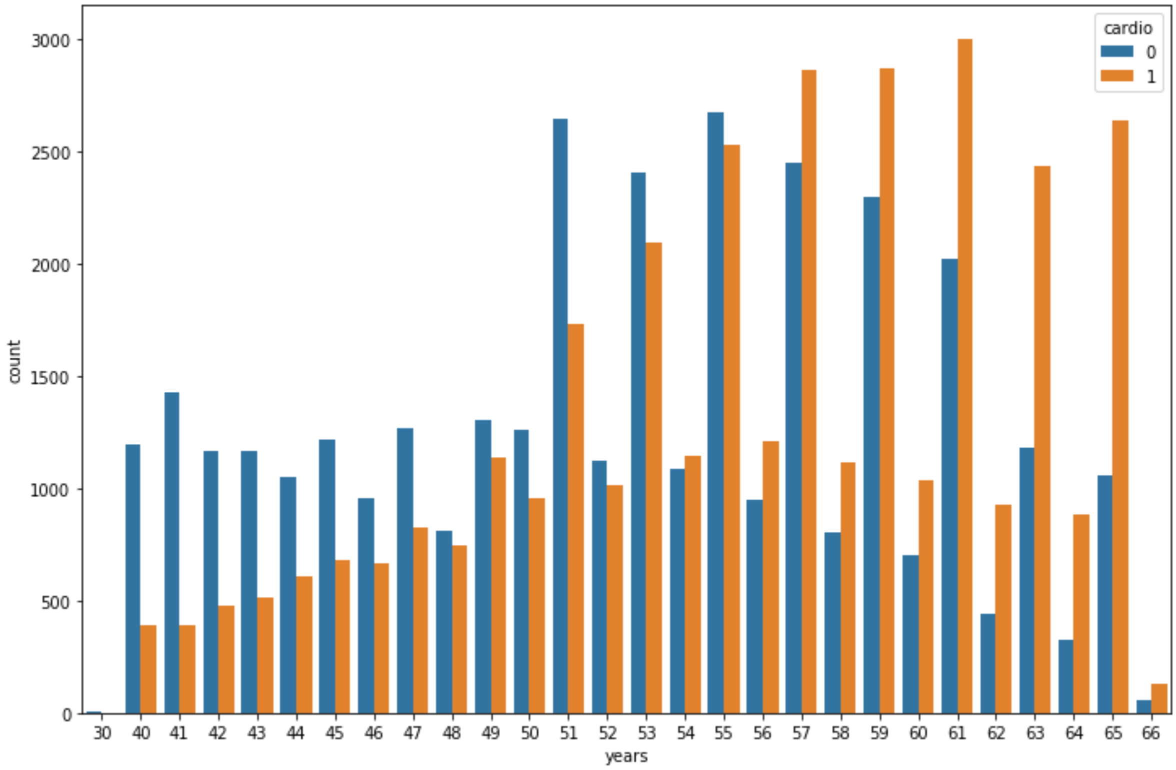
Out[10]:

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

Now plotting a graph to show the trend with CVDs and age

In [11]:
```
from matplotlib import rcParams
rcParams['figure.figsize'] = 12, 8 #To change figure size
sns.countplot(x = 'years', hue = 'cardio', data = dataset)
```

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1b1eb278>



As visible, people over the age of 55 show higher chances of being diaganosed with CVDs.

Now we find whether 1 stands for male or female. Assuming the male population is generally taller.

In [12]:
```
dataset.groupby('gender')['height'].mean()
```

Out[12]:
```
gender
1    161.355612
2    169.947895
Name: height, dtype: float64
```

As you can see, mean height of 2 is greater than 1.

Also, assuming that men generally weigh heavier than females,

In [13]:
```
dataset.groupby('gender')['weight'].mean()
```

Out[13]:
```
gender
1    72.565605
2    77.257307
Name: weight, dtype: float64
```

Mean weight of 2 > 1, hence our assumption is true. 1 -> Female 2 -> Male

Now, to check for height and weight: If you notice,

- Id's are irrelevant.
- The maximum height in the dataset is 250 cms.
- The maximum weight in the dataset is 200 kgs.
- The minimum height in the dataset is 55 cms.
- The minimum weight in the dataset is 10 kgs. Now considering that the minimum age is 30 years and maximum age is 66 years, it is highly likely that these cases are special and is better to treat them as outliers.

In [14]:
```
dataset.drop(['id'], axis = 1, inplace = True)
dataset.drop(dataset[(dataset['height'] > dataset['height'].quantil
e(0.975)) | (dataset['height'] < dataset['height'].quantile(0.025))
].index,inplace=True)
dataset.drop(dataset[(dataset['weight'] > dataset['weight'].quantil
e(0.975)) | (dataset['weight'] < dataset['weight'].quantile(0.025))
].index,inplace=True)
```

In [15]:
```
dataset.describe()
```

Out[15]:

	age	gender	height	weight	ap_hi	ap_l
count	63866.000000	63866.000000	63866.000000	63866.000000	63866.000000	63866.00000
mean	19472.641171	1.347806	164.497855	73.543564	128.815442	95.95330
std	2461.983315	0.476278	6.862322	11.720806	160.987785	186.28738
min	10798.000000	1.000000	150.000000	52.000000	-150.000000	-70.00000
25%	17679.250000	1.000000	160.000000	65.000000	120.000000	80.00000
50%	19705.000000	1.000000	165.000000	72.000000	120.000000	80.00000
75%	21323.000000	2.000000	169.000000	81.000000	140.000000	90.00000
max	23713.000000	2.000000	180.000000	106.000000	16020.000000	11000.00000

As you can now see, the minimum and maximum values for height and weight seems reasonable, after considering these values for over a range of 2.5%<= x <= 97.5%

Now coming to the blood pressures.

- We know that the diastolic blood pressure cannot exceed the systolic blood pressure.
- Blood pressure cannot be negative.

Using these constraints, we eliminate any outliers.

In [16]:
```
dataset.drop(dataset[(dataset['ap_hi'] > dataset['ap_hi'].quantile(0.975)) | (dataset['ap_hi'] < dataset['ap_hi'].quantile(0.025))].index,inplace=True)
dataset.drop(dataset[(dataset['ap_lo'] > dataset['ap_lo'].quantile(0.975)) | (dataset['ap_lo'] < dataset['ap_lo'].quantile(0.025))].index,inplace=True)
```

In [17]:
```
dataset.describe()
```

Out[17]:

	age	gender	height	weight	ap_hi	ap_l
count	60142.000000	60142.000000	60142.000000	60142.000000	60142.000000	60142.00000
mean	19468.719979	1.347311	164.554854	73.426805	125.770526	81.04630
std	2460.510296	0.476120	6.830174	11.614806	13.761847	8.23915
min	10798.000000	1.000000	150.000000	52.000000	100.000000	60.00000
25%	17677.250000	1.000000	160.000000	65.000000	120.000000	80.00000
50%	19705.000000	1.000000	165.000000	72.000000	120.000000	80.00000
75%	21321.000000	2.000000	169.000000	80.000000	135.000000	90.00000
max	23713.000000	2.000000	180.000000	106.000000	163.000000	100.00000

In [23]:
```
dataset.head()
```

Out[23]:

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

Now, we'll separate our dataset into two part to avoid any major changes in the main dataset.

In [24]:
```
X = dataset.drop(['age', 'cardio'], axis = 1)
y = dataset.iloc[:, -2]
```

Now it's time to separate the features and the target values and then split the dataset into training and test set.

In [25]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X, y, test_siz
e = 0.25, random_state = 0)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(45106, 11)
(15036, 11)
(45106,)
(15036,)
```

Now let us check to see if there's any missing data.

In [26]:
```
dataset.isnull().values.any()
```

Out[26]:
```
False
```

Since, there's no missing data, we proceed with Feature Scaling.

In [27]:
```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/data.
py:625: DataConversionWarning: Data with input dtype int64, float6
4 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/anaconda3/lib/python3.7/site-packages/sklearn/base.py:462: DataCo
nversionWarning: Data with input dtype int64, float64 were all con
verted to float64 by StandardScaler.
  return self.fit(X, **fit_params).transform(X)
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:4: Da
taConversionWarning: Data with input dtype int64, float64 were all
converted to float64 by StandardScaler.
  after removing the cwd from sys.path.
```

Let us create a class and list that will store our classifier's name, accuracy and the number of false negatives it generated.

We are considering the false negatives to be of high concern because it is better to diagnose a person with no CVD as a patient with CVD rather than the contrary.

In [28]:
```
class Classifier:
    def __init__(self, name, acc, falneg):
        self.name = name
        self.acc = acc
        self.falneg = falneg
    def __str__(self):
        return (f"Name of classifier: {self.name}\tAccuracy: {self.
acc}\tNo. of False Negatives: {self.falneg}")

clf_list = []
```

Now let's proceed with our models.

Starting off with SVM. Radial Basis Kernel.

In [29]:
```
from sklearn.svm import SVC
svc_clf = SVC(kernel = 'rbf', gamma = 'scale', random_state = 0)
svc_clf.fit(X_train, y_train)
svc_pred = svc_clf.predict(X_test)
from sklearn.metrics import accuracy_score
acc_svc = accuracy_score(y_test,svc_pred)
print(f"Accuracy for this model {acc_svc*100}")
```

```
Accuracy for this model 72.6855546687949
```

Accuracy for this model 72.6855546687949%

Let us see the confusion matrix for this kernel.

In [30]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, svc_pred)
print(cm)
```

```
[[6241 1497]
 [2610 4688]]
```

[[6241 1497] [2610 4688]] Is the confusion matrix.

As you can see, the number of false negatives is relatively quite high, which is a concern.

Let us add the classifier to the list.

In [31]:
```
clf_list.append(Classifier("SVC (rbf)", round(acc_svc*100, 4), cm[1
][0]))
print(clf_list[0])
```

```
Name of classifier: SVC (rbf)   Accuracy: 72.6856       No. of Fal
se Negatives: 2610
```

Now let us see for the polynomial kernel of SVC.

In [32]:
```
from sklearn.svm import SVC
svc_poly_clf = SVC(kernel = 'poly', degree = 3, gamma = 'scale', ra
ndom_state = 0)
svc_poly_clf.fit(X_train, y_train)
svc_poly_pred = svc_poly_clf.predict(X_test)
from sklearn.metrics import accuracy_score
acc_poly_svc = accuracy_score(y_test, svc_poly_pred)
print(f"Accuracy for this model {acc_poly_svc*100}")
```

```
Accuracy for this model 71.60148975791434
```

- Accuracy for this model 71.60814046288907% (degree = 3).
- Accuracy for this model 70.85661080074487% (degree = 4).
- Accuracy for this model 69.61957967544559% (degree = 5).

Therefore, as we increase the degree for the polynomial SVC, it's accuracy decreases.

Let us see the confusion matrix for this kernel.

In [33]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, svc_poly_pred)
print(cm)
```

```
[[6392 1346]
 [2924 4374]]
```

[[6392 1346] [2924 4374]] Is the confusion matrix.

As you can see, the number of false negatives is more than that of the rbf kernel, moreover the accuracy score of rbf is greater than that of the 3rd degree polynomial model.

Hence, let us move add the polynomial kernel to the list.

In [34]:
```
clf_list.append(Classifier("SVC (degree = 3)", round(acc_poly_svc*1
00, 4), cm[1][0]))
print(clf_list[1])
```

```
Name of classifier: SVC (degree = 3)    Accuracy: 71.6015
No. of False Negatives: 2924
```

Now, let us try the Naïve Bayes Model.

In [35]:
```
from sklearn.naive_bayes import GaussianNB
nb_clf = GaussianNB()
nb_clf.fit(X_train, y_train)
nb_pred = nb_clf.predict(X_test)
from sklearn.metrics import accuracy_score
acc_nb = accuracy_score(y_test,nb_pred)
print(f"Accuracy for this model {acc_nb*100}")
```

```
Accuracy for this model 71.32881085395051
```

Accuracy for this model 71.32881085395051%

Let us see the confusion matrix for this model.

In [36]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, nb_pred)
print(cm)
```

```
[[6261 1477]
 [2834 4464]]
```

[[6261 1477] [2834 4464]]

Is the confusion matrix.

Let us add this classifier to our list.

In [37]:
```
clf_list.append(Classifier("Naïve Bayes", round(acc_nb*100, 4), cm[
1][0]))
print(clf_list[2])
```

```
Name of classifier: Naïve Bayes Accuracy: 71.3288       No. of Fal
se Negatives: 2834
```

Now, let us try the Random Forest Classification Model.

In [38]:
```
from sklearn.ensemble import RandomForestClassifier
rf_clf = RandomForestClassifier(n_estimators = 100, criterion = 'en
tropy', random_state = 0)
rf_clf.fit(X_train, y_train)
rf_pred = rf_clf.predict(X_test)
from sklearn.metrics import accuracy_score
acc_rf = accuracy_score(y_test,rf_pred)
print(f"Accuracy for this model {acc_rf*100}")
```

```
Accuracy for this model 69.74594306996542
```

Accuracy for this model 69.74594306996542%, which is not as good.

Let us see the confusion matrix for this model.

In [39]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, rf_pred)
print(cm)
```

```
[[5573 2165]
 [2384 4914]]
```

[[5573 2165] [2384 4914]]

Is the confusion matrix.

If you observe the false negatives, it is quite good comapred to the other models.

Let us add this classifier to our list.

In [40]:
```
clf_list.append(Classifier("Random Forest", round(acc_rf*100, 4), c
m[1][0]))
print(clf_list[3])
```

```
Name of classifier: Random Forest       Accuracy: 69.7459
No. of False Negatives: 2384
```

Now, let us try the K-Nearest Neighbours Model

In [41]:
```
from sklearn.neighbors import KNeighborsClassifier
kn_clf = KNeighborsClassifier(n_neighbors = 150, metric = 'minkowsk
i', p = 2)
kn_clf.fit(X_train, y_train)
kn_pred = kn_clf.predict(X_test)
from sklearn.metrics import accuracy_score
acc_kn = accuracy_score(y_test,kn_pred)
print(f"Accuracy for this model {acc_kn*100}")
```

```
Accuracy for this model 72.29316307528599
```

Accuracy for this model 72.29316307528599

Let us see the confusion matrix for this model.

In [42]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, kn_pred)
print(cm)
```

```
[[6235 1503]
 [2663 4635]]
```

[[6235 1503] [2663 4635]]

Is the confusion matrix.

The number of false negatives for this model is somewhat average.

Let us add this classifier to our list.

In [43]:
```
clf_list.append(Classifier("K-NN", round(acc_kn*100, 4), cm[1][0]))
print(clf_list[4])
```

```
Name of classifier: K-NN        Accuracy: 72.2932       No. of Fal
se Negatives: 2663
```

Now, let us try the Logistic Regression model.

In [44]:
```
from sklearn.linear_model import LogisticRegression
lr_clf = LogisticRegression(random_state = 0, solver = 'liblinear',
multi_class = 'ovr')
lr_clf.fit(X_train, y_train)
lr_pred = lr_clf.predict(X_test)
from sklearn.metrics import accuracy_score
acc_lr = accuracy_score(y_test, lr_pred)
print(f"Accuracy for this model {acc_lr*100}")
```

```
Accuracy for this model 72.41952646980579
```

Accuracy for this model 72.41952646980579%

Let us see the confusion matrix for this model.

In [45]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, lr_pred)
print(cm)
```

```
[[6126 1612]
 [2535 4763]]
```

[[6126 1612] [2535 4763]]

Is the confusion matrix.

The number of false negatives for this model is somewhat decent.

Let us add this classifier to our list.

In [46]:
```
clf_list.append(Classifier("Logistic Regression", round(acc_lr*100,
4), cm[1][0]))
print(clf_list[5])
```

```
Name of classifier: Logistic Regression Accuracy: 72.4195
No. of False Negatives: 2535
```

Let us take a look at the scores of all our models.

In [47]:
```
for model in clf_list:
    print(model)
```

```
Name of classifier: SVC (rbf)   Accuracy: 72.6856       No. of Fal
se Negatives: 2610
Name of classifier: SVC (degree = 3)    Accuracy: 71.6015
No. of False Negatives: 2924
Name of classifier: Naïve Bayes Accuracy: 71.3288       No. of Fal
se Negatives: 2834
Name of classifier: Random Forest       Accuracy: 69.7459
No. of False Negatives: 2384
Name of classifier: K-NN        Accuracy: 72.2932       No. of Fal
se Negatives: 2663
Name of classifier: Logistic Regression Accuracy: 72.4195
No. of False Negatives: 2535
```