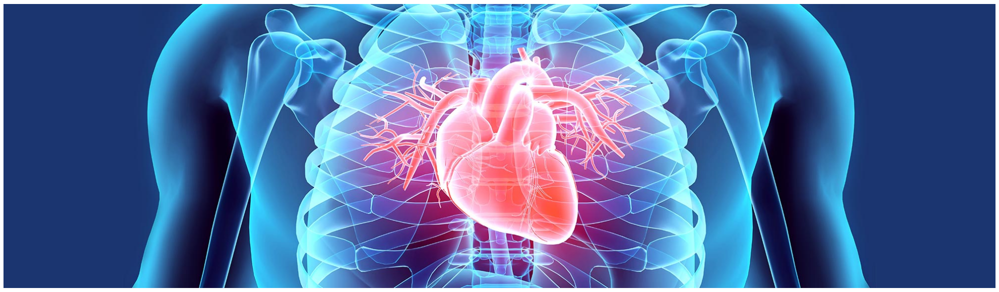


Introduction

The Cardiovascular Disease dataset is a collection of health-related information used for analyzing and predicting cardiovascular diseases. It encompasses diverse data such as age, gender, blood pressure, cholesterol levels, and lifestyle factors. This dataset serves as a valuable resource for researchers and healthcare professionals to better understand and mitigate the risks associated with cardiovascular diseases.

1 import Necessary Library

In [115…

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

2 import Dataset

In [116…

```
df = pd.read_csv("/kaggle/input/cardiovascular-disease-dataset/cardio_train.csv
```

3 Data Analysis

In [117…

```
df.head()
```

Out[117…

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	ac
0	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	
2	2	18857	1	165	64.0	130	70	3	1	0	0	
3	3	17623	2	169	82.0	150	100	1	1	0	0	
4	4	17474	1	156	56.0	100	60	1	1	0	0	

In [118…

```
df.tail()
```

Out[118…

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

In [119…

```
df.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
```

In [120…

```
df.describe()
```

Out[120…

	id	age	gender	height	weight	ap
count	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000
mean	49972.419900	19468.865814	1.349571	164.359229	74.205690	128.817
std	28851.302323	2467.251667	0.476838	8.210126	14.395757	154.011
min	0.000000	10798.000000	1.000000	55.000000	10.000000	-150.000
25%	25006.750000	17664.000000	1.000000	159.000000	65.000000	120.000
50%	50001.500000	19703.000000	1.000000	165.000000	72.000000	120.000
75%	74889.250000	21327.000000	2.000000	170.000000	82.000000	140.000
max	99999.000000	23713.000000	2.000000	250.000000	200.000000	16020.000

In [121…

```
df.corr()
```

Out[121…

	id	age	gender	height	weight	ap_hi	ap_lo
id	1.000000	0.003457	0.003502	-0.003038	-0.001830	0.003356	-0.002529
age	0.003457	1.000000	-0.022811	-0.081515	0.053684	0.020764	0.017647

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

In [122…

```
df.columns
```

Out[122…

```
Index(['id', 'age', 'gender', 'height', 'weight', 'ap_hi', 'ap_lo',
       'cholesterol', 'gluc', 'smoke', 'alco', 'active', 'cardio'],
      dtype='object')
```

In [123…

```
df["cardio"].value_counts()
```

Out[123…

```
cardio
0    35021
1    34979
Name: count, dtype: int64
```

4 Data cleaning and Preprocessing:

In [124…

```
df.head()
```

Out[124…

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	ac
0	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	
2	2	18857	1	165	64.0	130	70	3	1	0	0	
3	3	17623	2	169	82.0	150	100	1	1	0	0	
4	4	17474	1	156	56.0	100	60	1	1	0	0	

In [125…

```
df['age']=(df['age']/365).round(0)
```

In [126…

```
df.head()
```

Out[126…

id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	activ

0	0	50.0	2	168	62.0	110	80	1	1	0	0
1	1	55.0	1	156	85.0	140	90	3	1	0	0
2	2	52.0	1	165	64.0	130	70	3	1	0	0
3	3	48.0	2	169	82.0	150	100	1	1	0	0
4	4	48.0	1	156	56.0	100	60	1	1	0	0

In [127…

```
df = df.drop(['id'], axis=1)
```

In [128…

```
df.head()
```

Out[128…

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active
0	50.0	2	168	62.0	110	80	1	1	0	0	1
1	55.0	1	156	85.0	140	90	3	1	0	0	1
2	52.0	1	165	64.0	130	70	3	1	0	0	0
3	48.0	2	169	82.0	150	100	1	1	0	0	1
4	48.0	1	156	56.0	100	60	1	1	0	0	0

In [129…

```
df.isnull().sum()
```

Out[129…

```
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 [130…

```
df.isnull().values.any()
```

Out[130…

```
False
```

In [131…

```
df.isna().sum()
```

Out[131…

```
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
```

5| Data visualisation

EDA(Exploratory Data Analysis)

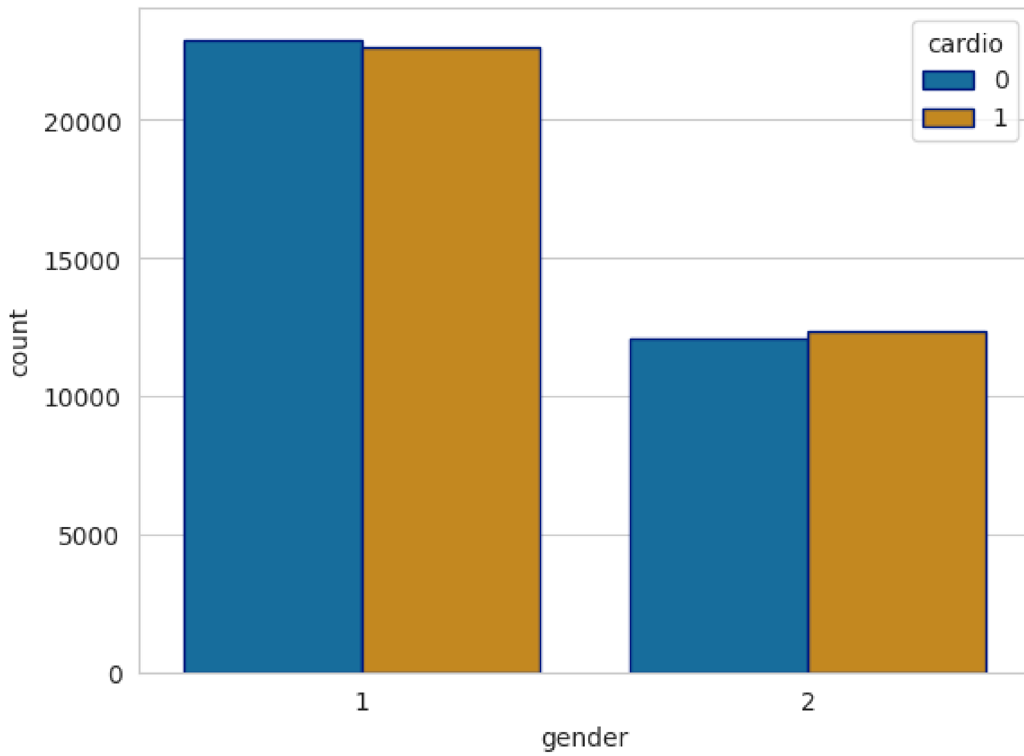
5.1 Countplot

In [132…

```python
sns.countplot(x="gender", hue="cardio", data=df, palette="colorblind", edgecolo
```

Out[132…

```
<Axes: xlabel='gender', ylabel='count'>
```



In [133…

```python
df['gender'].value_counts()
```

Out[133…

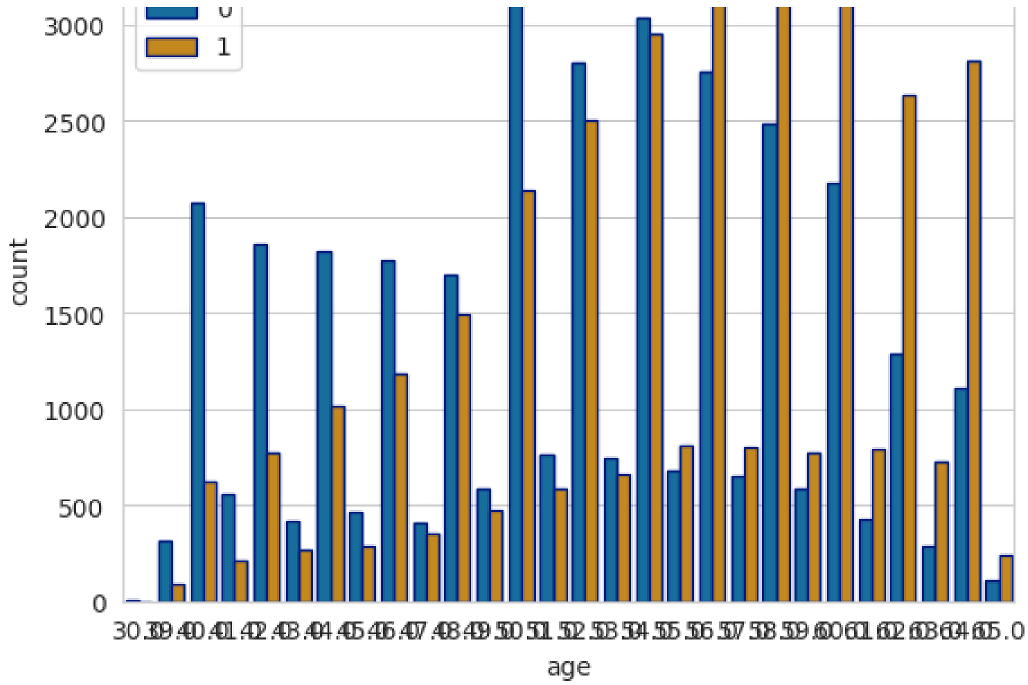
```
gender
1    45530
2    24470
Name: count, dtype: int64
```

In [134…

```python
sns.countplot(x="age", hue="cardio", data=df, palette="colorblind", edgecolor=s
```

Out[134…

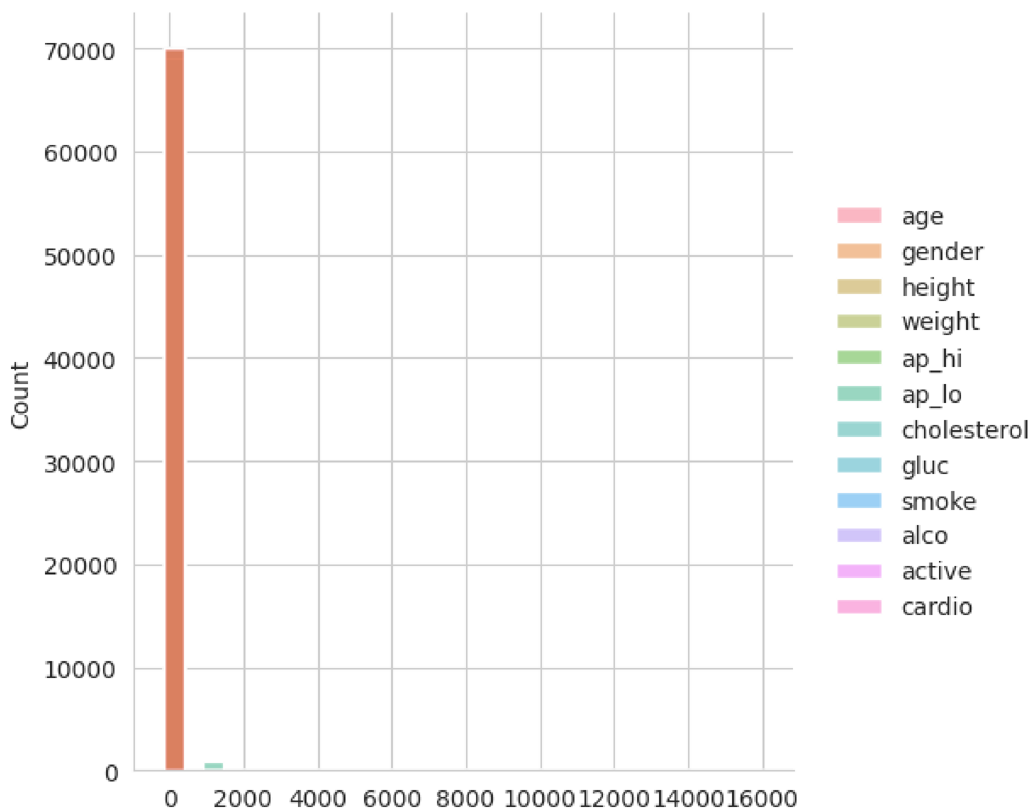
```
<Axes: xlabel='age', ylabel='count'>
```



5.2 displot

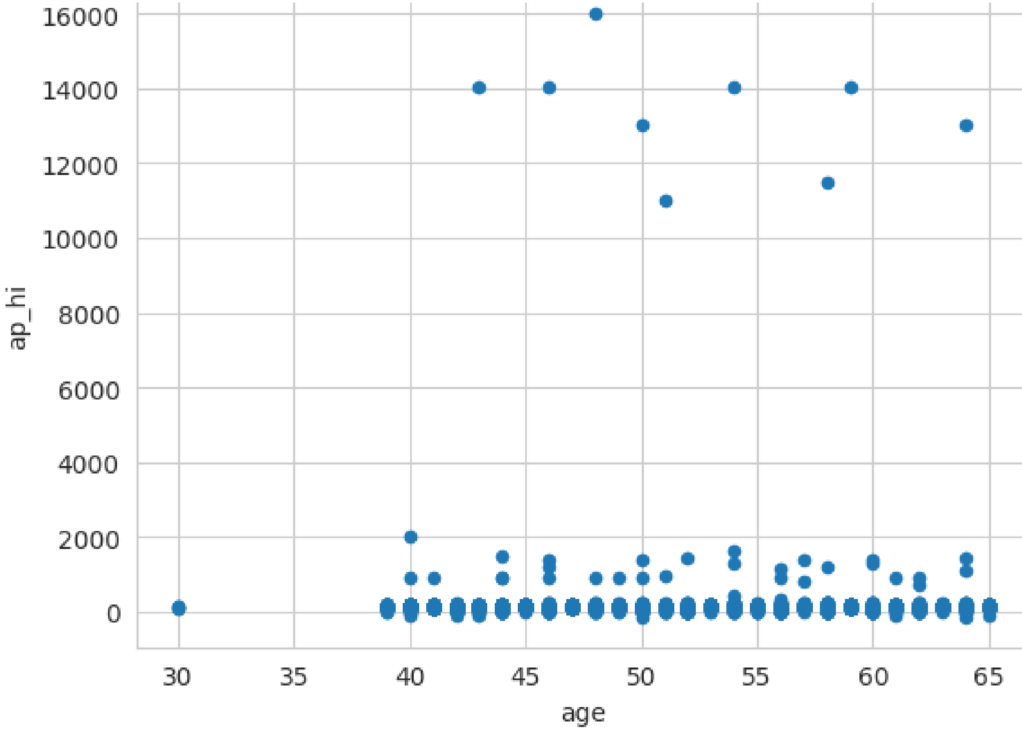
In [135…

```
sns.displot(df, kde=False, bins=30)
plt.show()
```



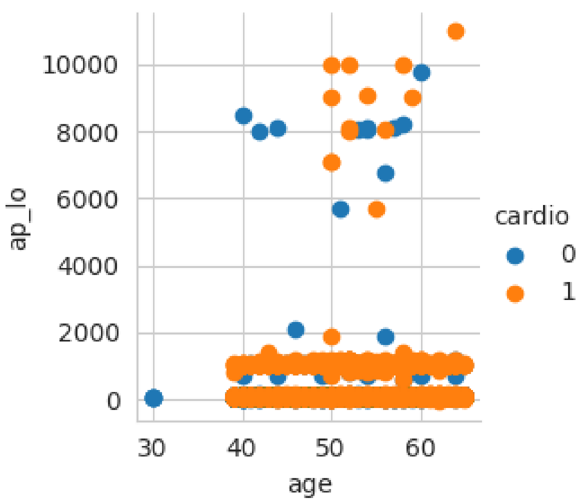
In [136…

```
df.plot(kind='scatter', x='age', y='ap_hi')
plt.show()
```



In [137…

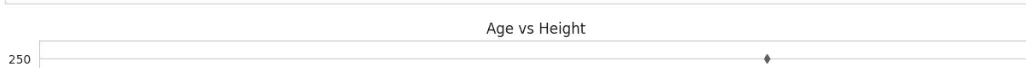
```
sns.set_style("whitegrid");
sns.FacetGrid(df, hue="cardio").map(plt.scatter, "age", "ap_lo").add_legend();
plt.show();
```

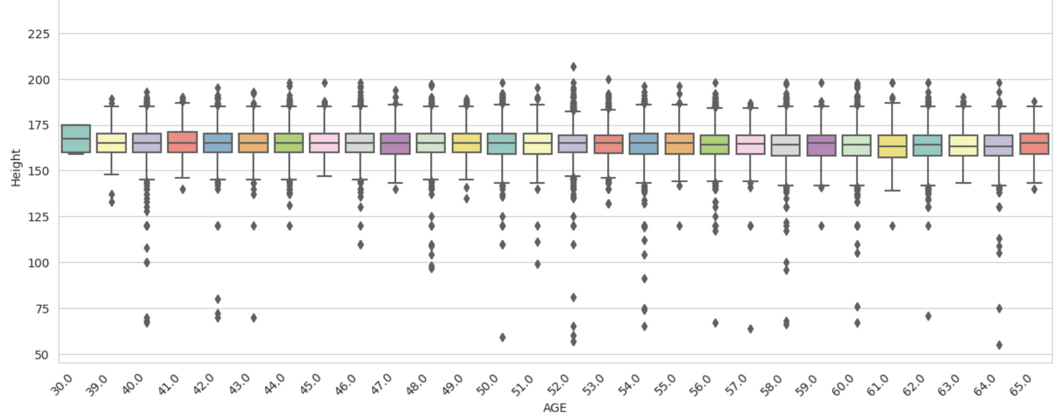


5.3 boxplot

In [138…

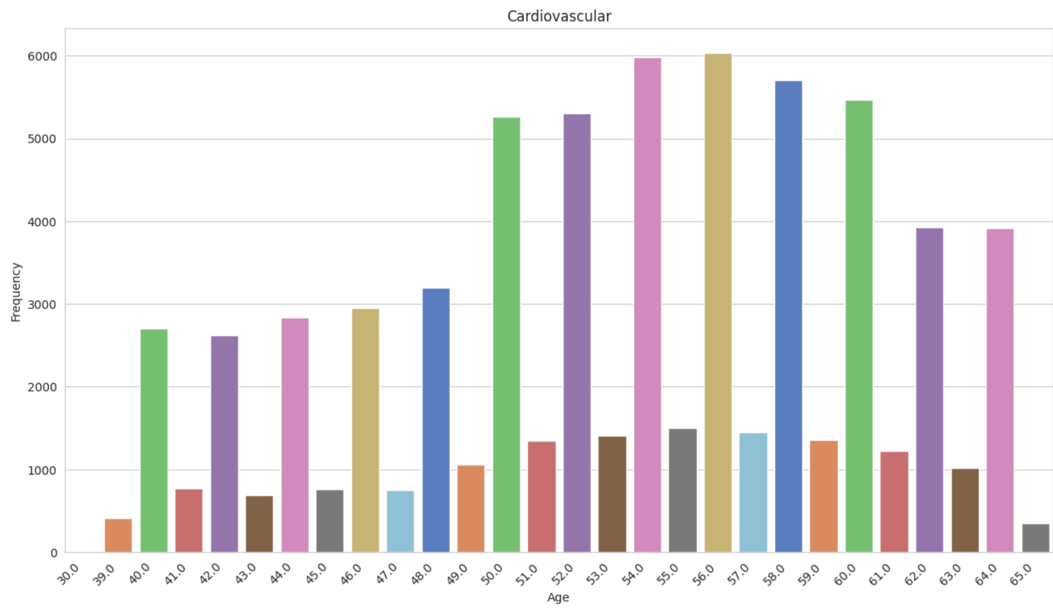
```
# sepal_length vs sepal_width boxplot

plt.figure(figsize=(15, 6))
sns.boxplot(x='age', y='height', data=df, palette='Set3')
plt.title('Age vs Height')
plt.xlabel('AGE')
plt.ylabel('Height')
plt.xticks(rotation=45, ha='right')
plt.show()
```





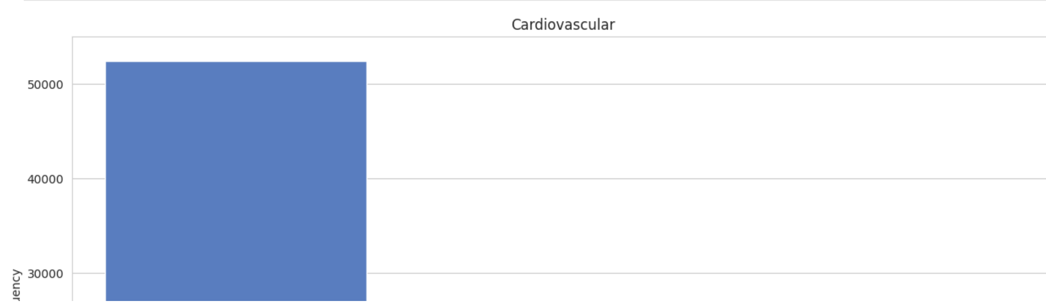
In [139…

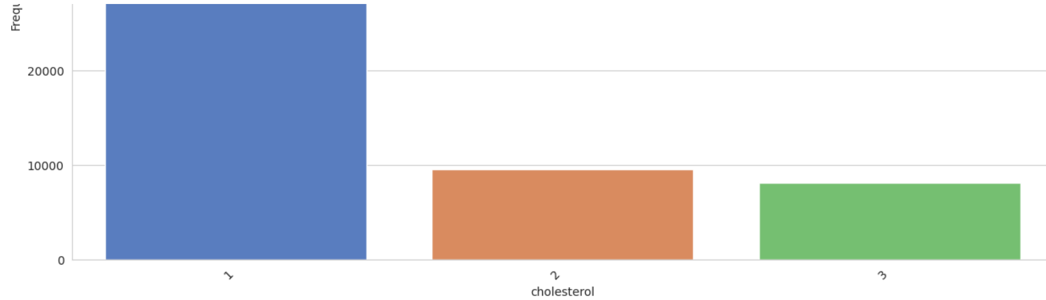
```
# sepal_length
plt.figure(figsize=(15, 8))
sns.countplot(x='age', data=df, palette='muted')
plt.title('Cardiovascular')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.xticks(rotation=45, ha='right')
plt.show()
```



In [140…

```
# sepal_length
plt.figure(figsize=(15, 8))
sns.countplot(x='cholesterol', data=df, palette='muted')
plt.title('Cardiovascular')
plt.xlabel('cholesterol')
plt.ylabel('Frequency')
plt.xticks(rotation=45, ha='right')
plt.show()
```

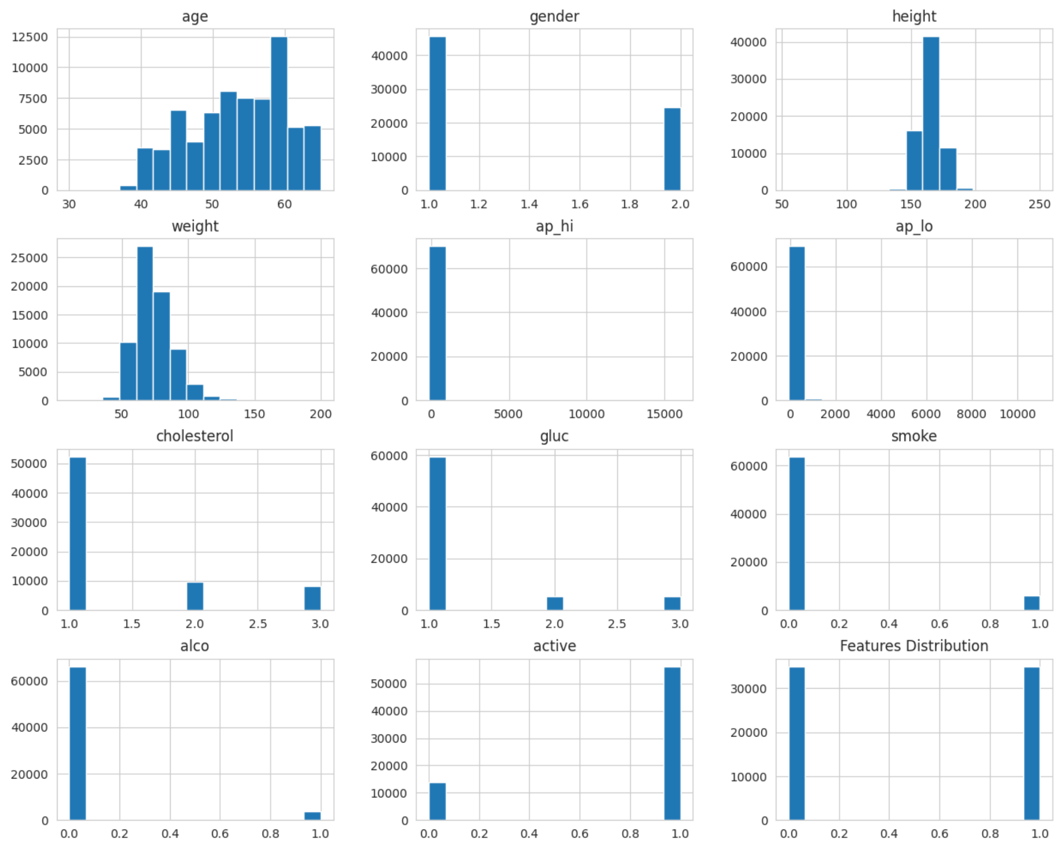




5.4 hist plot

In [141…

```
df.hist(figsize=(15,12),bins = 15)
plt.title("Features Distribution")
plt.show()
```



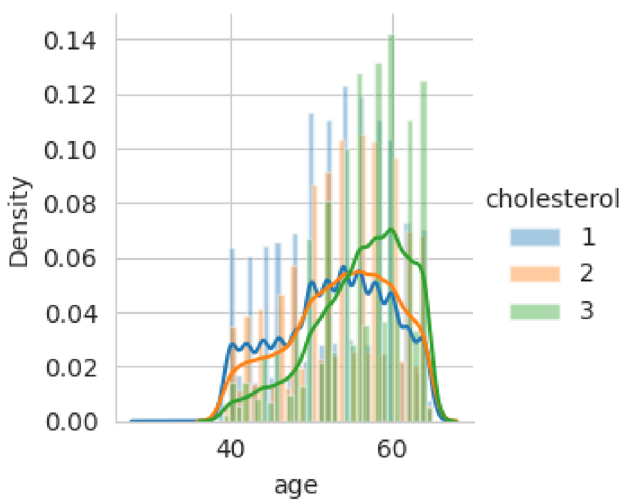
5.5 dens_plot

In [142…

```
def dens_plot(features,class_var):
    #adding the white grid style
    sns.set_style(style="whitegrid")
    #adding datapoint colour and size
    sns.FacetGrid(data=df, hue=class_var)\
        .map(sns.distplot,features)\
        .add_legend()

dens_plot("age","cholesterol")
plt.show()
```

/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:848: UserWarning:

```
/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:848: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  func(*plot_args, **plot_kwargs)
/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:848: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  func(*plot_args, **plot_kwargs)
/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:848: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  func(*plot_args, **plot_kwargs)
```



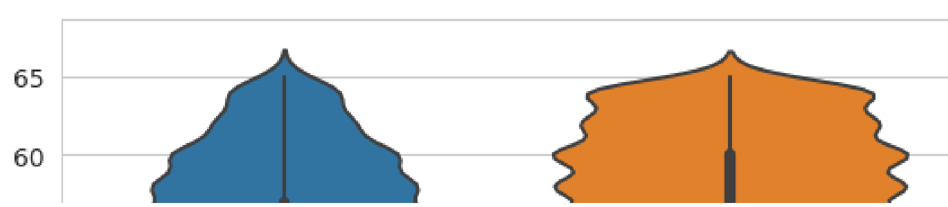
5.6 violinplot

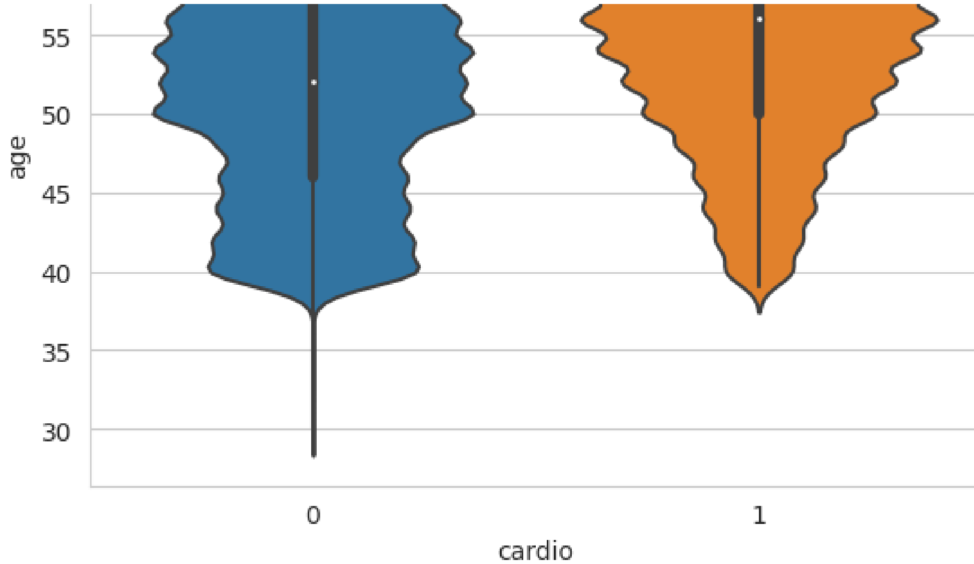
In [143…

```python
sns.violinplot(x="cardio",y="age", data=df, size = 8)
```

Out[143…

```
<Axes: xlabel='cardio', ylabel='age'>
```

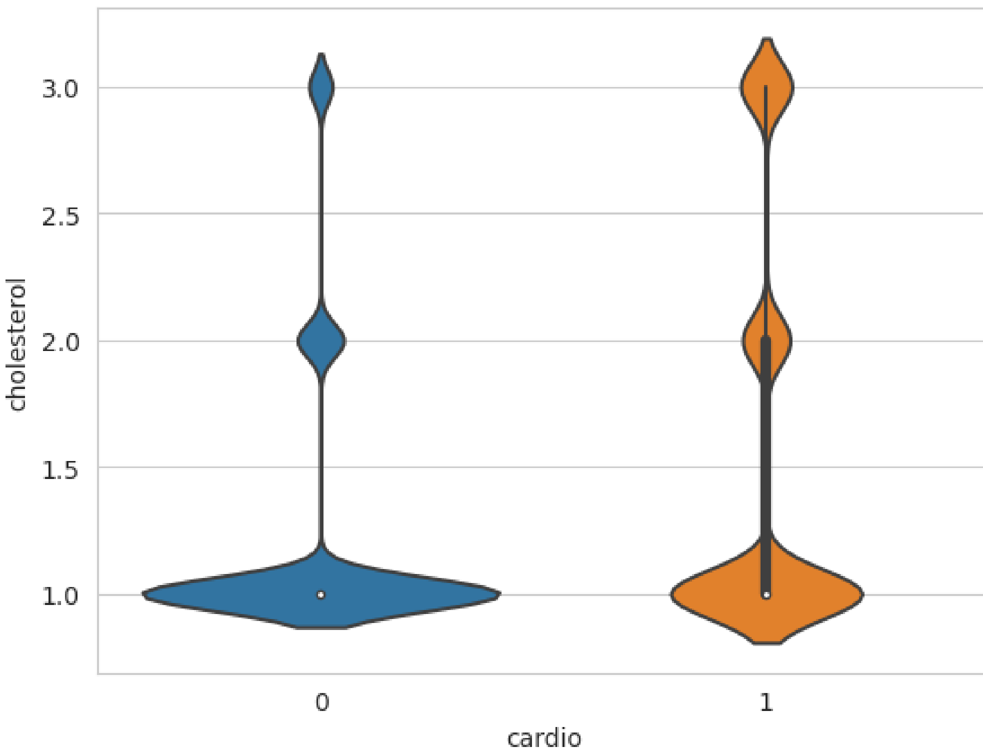




In [144…

```
sns.violinplot(x="cardio",y="cholesterol", data=df, size = 8)
```

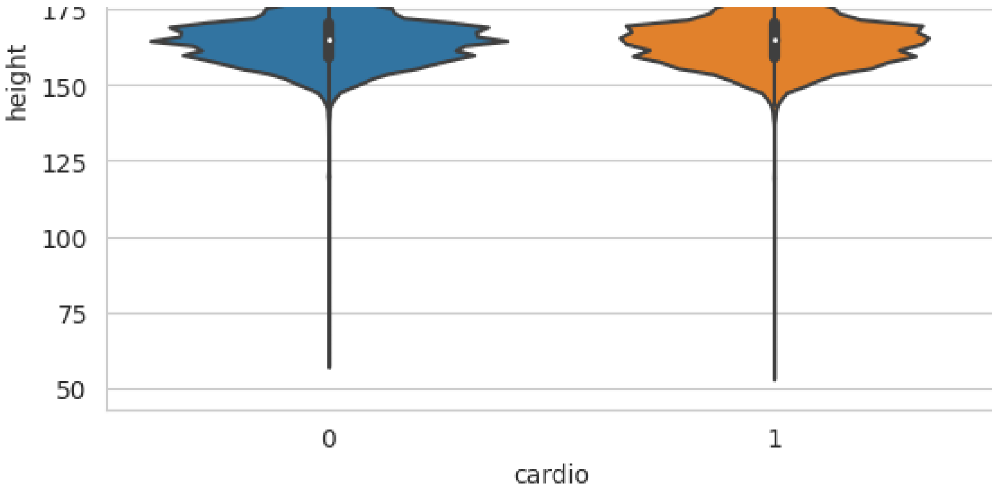
Out[144… <Axes: xlabel='cardio', ylabel='cholesterol'>



In [145…

```
sns.violinplot(x="cardio",y="height", data=df, size = 8)
```

Out[145… <Axes: xlabel='cardio', ylabel='height'>

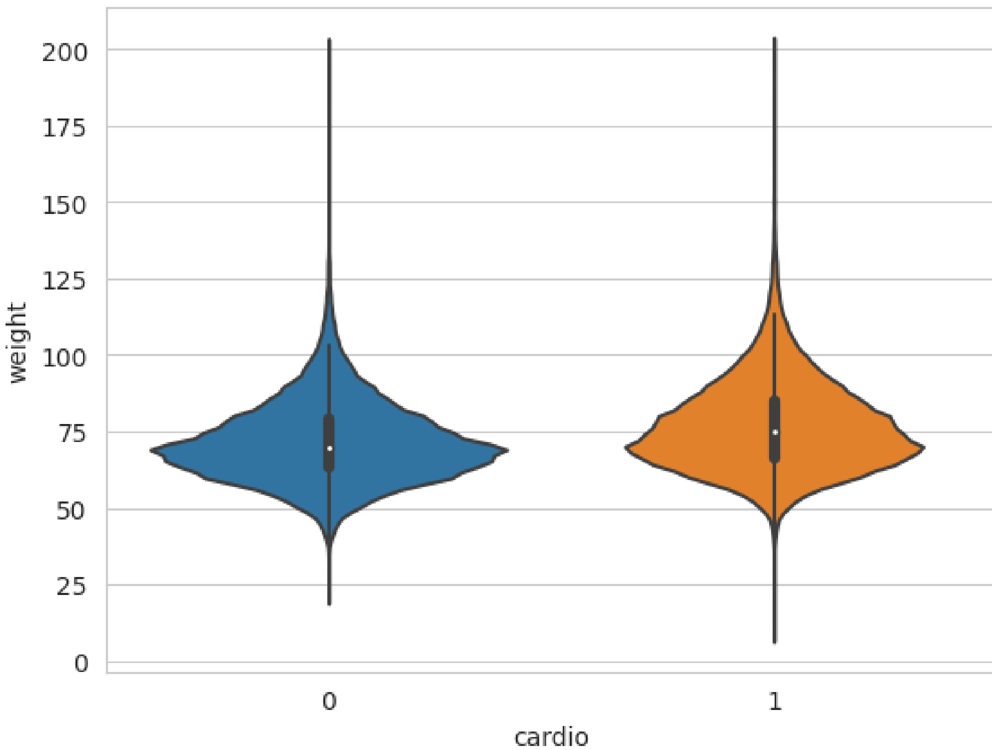


In [146…

```
sns.violinplot(x="cardio",y="weight", data=df, size = 8)
```

Out[146…

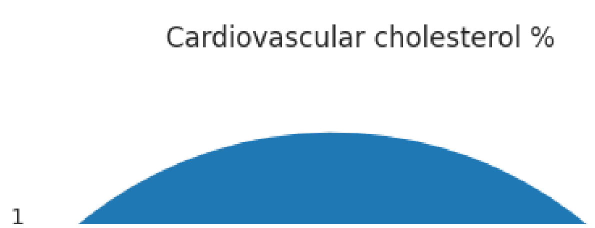
```
<Axes: xlabel='cardio', ylabel='weight'>
```

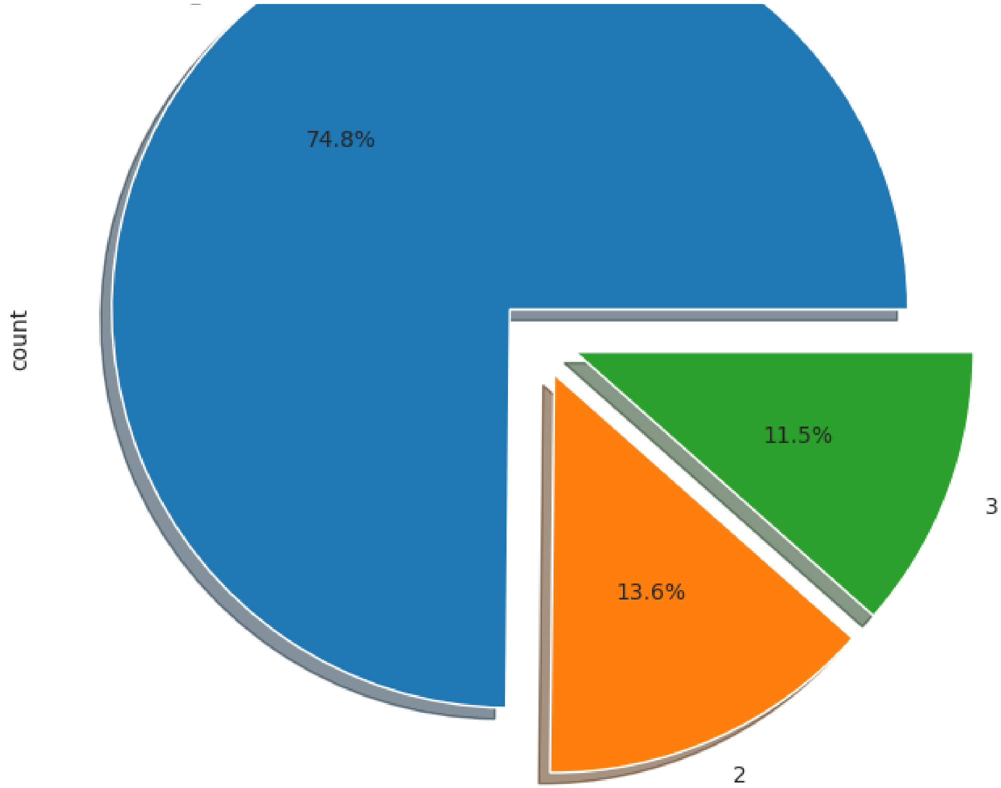


5.7 Pie Plot

In [147…

```
ax=plt.subplots(1,1,figsize=(10,8))
df['cholesterol'].value_counts().plot.pie(explode=[0.1,0.1,0.1],autopct='%1.1f%
plt.title("Cardiovascular cholesterol %")
plt.show()
```





In [148…

```
df.head()
```

Out[148…

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active
0	50.0	2	168	62.0	110	80	1	1	0	0	1
1	55.0	1	156	85.0	140	90	3	1	0	0	1
2	52.0	1	165	64.0	130	70	3	1	0	0	0
3	48.0	2	169	82.0	150	100	1	1	0	0	1
4	48.0	1	156	56.0	100	60	1	1	0	0	0

6 | Split the Dataset

In [149…

```
df = df[:10000]
```

In [150…

```
from sklearn.model_selection import train_test_split
```

In [151…

```
x = df.iloc[:,:-1]
```

In [152…

```
x.head()
```

Out[152…

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active

0	50.0	2	168	62.0	110	80	1	1	0	0	1
1	55.0	1	156	85.0	140	90	3	1	0	0	1
2	52.0	1	165	64.0	130	70	3	1	0	0	0
3	48.0	2	169	82.0	150	100	1	1	0	0	1
4	48.0	1	156	56.0	100	60	1	1	0	0	0

In [153…
```
y = df.iloc[:,11]
```

In [154…
```
y.head()
```

Out[154…
```
0    0
1    1
2    1
3    1
4    0
Name: cardio, dtype: int64
```

In [155…
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.2, randon
```

In [156…
```
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

Out[156…
```
((8000, 11), (2000, 11), (8000,), (2000,))
```

7 | PCA (Principal Component Analysis)

In [157…
```
from sklearn.decomposition import PCA
```

In [158…
```
pca = PCA(n_components=2)
```

In [159…
```
X_pca = pca.fit_transform(x)
```

In [160…
```
X_pca[0]
```

Out[160…
```
array([-18.27717182, -17.81552921])
```

In [161…
```
pca.explained_variance_ratio_
```

Out[161…
```
array([0.75452656, 0.23975076])
```

In [ ]:
```

```

Machine Learning Algorithm

(1). KNN

In [162…
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

In [163…
```
knn_classifier = KNeighborsClassifier(n_neighbors=3)
```

In [164…
```
knn_classifier.fit(x_train, y_train)
```

Out[164…
```
KNeighborsClassifier(n_neighbors=3)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

- 🔥 Test accuracy KNN

In [165…
```
y_pred = knn_classifier.predict(x_test)
```

In [166…
```
accuracy = accuracy_score(y_test, y_pred)
```

In [167…
```
accuracy
```

Out[167…
```
0.673
```

- 🔥 Training accuracy KNN

In [168…
```
y_pred_train = knn_classifier.predict(x_train)
```

In [169…
```
accuracy = accuracy_score(y_pred_train, y_train)
```

In [170…
```
accuracy
```

Out[170…
```
0.80775
```

(2). Naive Bayes classifier

In [171…
```
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import MultinomialNB

from sklearn import metrics
```

In [172…
```
# GaussianNB
```

In [173…
```
G_classifier = GaussianNB()
```

In [174…
```
G_classifier.fit(x_train, y_train)
```

Out[174…
```
GaussianNB()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

- 🔥 Test accuracy (Naive Bayes) GaussianNB

In [175…
```
predictions_G = G_classifier.predict(x_test)
```

In [176…
```
accuracy = metrics.accuracy_score(y_test, predictions_G)
```

In [177…
```
accuracy
```

Out[177…
```
0.5965
```

- 🔥 Training accuracy (Naive Bayes) GaussianNB

In [178…
```
predictions_G = G_classifier.predict(x_train)
```

In [179…
```
accuracy = metrics.accuracy_score(y_train, predictions_G)
```

In [180…
```
accuracy
```

Out[180…
```
0.583
```

In [181…
```
# BernoulliNB
```

In [182…
```
B_classifier = BernoulliNB()
```

In [183…
```
B_classifier.fit(x_train, y_train)
```

Out[183…
```
BernoulliNB()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

- 🔥 Test accuracy (Naive Bayes) BernoulliNB

In [184

In [184…
```
predictions_B = B_classifier.predict(x_test)
```

In [185…
```
accuracy_B = metrics.accuracy_score(y_test, predictions_B)
```

In [186…
```
accuracy_B
```

Out[186…
```
0.5185
```

- 🔥 Training accuracy (Naive Bayes) BernoulliNB

In [187…
```
predictions_B = B_classifier.predict(x_train)
```

In [188…
```
accuracy_B = metrics.accuracy_score(y_train, predictions_B)
```

In [189…
```
accuracy_B
```

Out[189…
```
0.51275
```

(3) Decision Tree

In [190…
```
from sklearn.tree import DecisionTreeClassifier
```

In [191…
```
clf = DecisionTreeClassifier()
```

In [192…
```
clf.fit(x_train, y_train)
```

Out[192…
```
DecisionTreeClassifier()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [193…
```
train_predictions = clf.predict(x_train)

train_accuracy3 = accuracy_score(y_train, train_predictions)
```

In [194…
```
test_predictions = clf.predict(x_test)

test_accuracy3 = accuracy_score(y_test, test_predictions)
```

In [195…
```
print(f"Training Accuracy: {train_accuracy3}")
print(f"Testing Accuracy: {test_accuracy3}")
```

```
Training Accuracy: 0.994125
Testing Accuracy: 0.636
```

(4) Random Forest

(4). Random Forest

In [196…
```
from sklearn.ensemble import RandomForestClassifier
```

In [197…
```
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
```

In [198…
```
rf_classifier.fit(x_train, y_train)
```

Out[198…
```
RandomForestClassifier(random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [199…
```
train_predictions = rf_classifier.predict(x_train)

train_accuracy4 = accuracy_score(y_train, train_predictions)
```

In [200…
```
test_predictions = rf_classifier.predict(x_test)

test_accuracy4 = accuracy_score(y_test, test_predictions)
```

In [201…
```
print(f"Training Accuracy: {train_accuracy4}")
print(f"Testing Accuracy: {test_accuracy4}")
```

```
Training Accuracy: 0.994125
Testing Accuracy: 0.7125
```

(5). Boosting Algorithm

In [202…
```
from sklearn.ensemble import AdaBoostClassifier
```

In [203…
```
base_classifier = DecisionTreeClassifier(max_depth=3)
```

In [204…
```
adaboost_classifier = AdaBoostClassifier(base_classifier, n_estimators=50, rar
```

In [205…
```
adaboost_classifier.fit(x_train, y_train)
```

Out[205…
```
AdaBoostClassifier(estimator=DecisionTreeClassifier(max_depth=3),
                   random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [206…
```
train_predictions = adaboost_classifier.predict(x_train)

train_accuracy5 = accuracy_score(y_train, train_predictions)
```

In [207…

```
test_predictions = adaboost_classifier.predict(x_test)

test_accuracy5 = accuracy_score(y_test, test_predictions)
```

In [208…

```
print(f"Training Accuracy: {train_accuracy5}")
print(f"Testing Accuracy: {test_accuracy5}")
```

```
Training Accuracy: 0.759125
Testing Accuracy: 0.724
```

(6).SVM

In [209…

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

In [210…

```
scaler = StandardScaler()
X_train = scaler.fit_transform(x_train)
X_test = scaler.transform(x_test)
```

In [211…

```
svm_classifier = SVC(kernel='linear', C=1.0)
```

In [212…

```
svm_classifier.fit(x_train, y_train)
```

Out[212…

```
SVC(kernel='linear')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [213…

```
train_predictions = svm_classifier.predict(X_train)

train_accuracy6 = accuracy_score(y_train, train_predictions)
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/base.py:439: UserWarning: X does
not have valid feature names, but SVC was fitted with feature names
  warnings.warn(
```

In [214…

```
test_predictions = svm_classifier.predict(x_test)

test_accuracy6 = accuracy_score(y_test, test_predictions)
```

In [215…

```
print(f"Training Accuracy: {train_accuracy6}")
print(f"Testing Accuracy: {test_accuracy6}")
```

```
Training Accuracy: 0.496
Testing Accuracy: 0.7255
```

(7). Logistic Regression

In [216

In [216...
```
from sklearn import linear_model
```

In [217...
```
lrg = linear_model.LogisticRegression()
```

In [218...
```
lrg.fit(x_train, y_train)
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: C
onvergenceWarning: 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
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n
  n_iter_i = _check_optimize_result(
```

Out[218...
```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [219...
```
train_predictions = lrg.predict(X_train)

train_accuracy7 = accuracy_score(y_train, train_predictions)
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/base.py:439: UserWarning: X does
not have valid feature names, but LogisticRegression was fitted with feature nam
es
  warnings.warn(
```

In [220...
```
test_predictions = lrg.predict(X_test)

test_accuracy7 = accuracy_score(y_test, test_predictions)
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/base.py:439: UserWarning: X does
not have valid feature names, but LogisticRegression was fitted with feature nam
es
  warnings.warn(
```

In [221...
```
print(f"Training Accuracy: {train_accuracy7}")
print(f"Testing Accuracy: {test_accuracy7}")
```

```
Training Accuracy: 0.578
Testing Accuracy: 0.5815
```

(9).Gradient Boosting Machines (GBM)

In [222...
```
from sklearn.ensemble import GradientBoostingClassifier
```

In [223...
```
model = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_de
```

In [224...
```
model.fit(X_train, y_train)
```

Out[224…
```
GradientBoostingClassifier(random_state=42)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [225…
```
train_predictions = model.predict(X_train)

train_accuracy9 = accuracy_score(y_train, train_predictions)
```

In [226…
```
test_predictions = model.predict(X_test)

test_accuracy9 = accuracy_score(y_test, test_predictions)
```

In [227…
```
print(f"Training Accuracy: {train_accuracy9}")
print(f"Testing Accuracy: {test_accuracy9}")
```
```
Training Accuracy: 0.74425
Testing Accuracy: 0.737
```

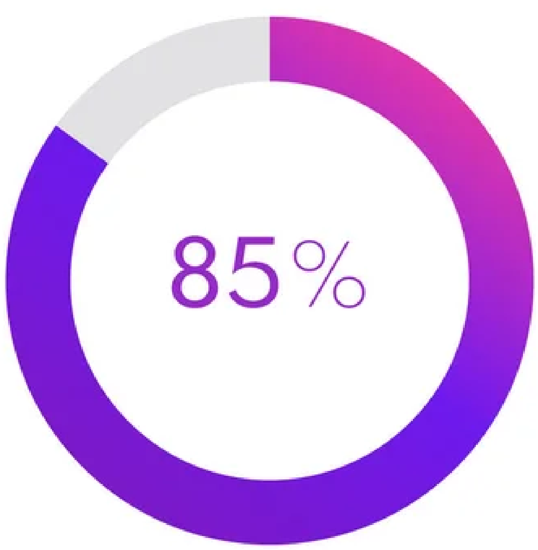
Random Forest Algorithm is the best accuracy

(Random Forest)

Training Accuracy: 0.994125

Testing Accuracy: 0.7125

** mean Accuracy : 0.85



10 | Hierarchical Clustering

In [228…

```
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster
from sklearn.preprocessing import StandardScaler
```

In [229…

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(x)
```

In [230…

```
linkage_matrix = linkage(X_scaled, method='ward')
```

In [233…

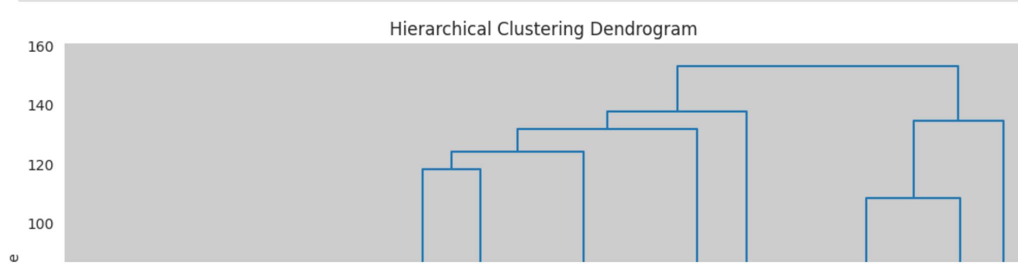
```
df.head()
```

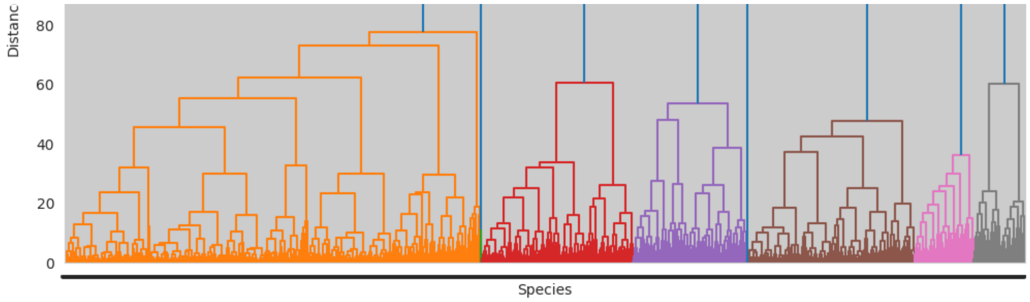
Out[233…

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active
0	50.0	2	168	62.0	110	80	1	1	0	0	1
1	55.0	1	156	85.0	140	90	3	1	0	0	1
2	52.0	1	165	64.0	130	70	3	1	0	0	0
3	48.0	2	169	82.0	150	100	1	1	0	0	1
4	48.0	1	156	56.0	100	60	1	1	0	0	0

In [234…

```
plt.figure(figsize=(12, 6))
dendrogram(linkage_matrix, labels=df['cardio'].values, orientation='top', dist
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('cardio')
plt.ylabel('Distance')
plt.show()
```





In [ ]: