# Cardiocvascular Disease Prediction

## DataSource : Kaggle[sulianova]

**Team** -

[BL.EN.U4AIE19007] *Apoorva Mani*

[BL.EN.U4AIE19010] *Bhuvanashree Murugadoss*

[BL.EN.U4AIE19027] *Karna Sai Nikhilesh Reddy*

# Importing Dependencies

In [18]:

```
import sys # Not Required
import warnings
import os
import pickle
import numpy as np
import pandas as pd
import matplotlib as mpl
from matplotlib import pyplot as plt
from pandas.plotting import scatter_matrix
import seaborn as sns

from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split, StratifiedShuffleSplit, cross_val
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, pl

# Machine Learning Algorithms
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import RandomForestClassifier

# Ignoring Warnings
warnings.filterwarnings(action='ignore')

# Fixing matplotlib inline and label sizes
%matplotlib inline
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
```

# File Locations for Images, Dataset, Pickles and Models

In [19]:

```
# Root Directory
PROJECT_ROOT_DIR = os.getcwd()

# Dataset Directory
DATASET_NAME = 'cardiovascular-disease-dataset.csv'
DATASET_DIR = 'datasets'
DATASET_PATH = os.path.join(PROJECT_ROOT_DIR, DATASET_DIR)


def load_dataset(path=DATASET_PATH, filename=DATASET_NAME, sep=';'):
    dataset_location = os.path.join(path, filename)
    return pd.read_csv(dataset_location, sep)


# Pickle and Model Directory
PM_DIR = 'Pickles_And_Models'
PM_PATH = os.path.join(PROJECT_ROOT_DIR, PM_DIR)
os.makedirs(PM_PATH, exist_ok=True)

def save_object(object_ , pickle_name, pm_path = PM_PATH):
    path = os.path.join(pm_path, pickle_name)
    pickle.dump(object_, open(path, 'wb'))
    print('Saving Pickle', path)

def load_object(pickle_name, pm_path = PM_PATH):
    path = os.path.join(pm_path, pickle_name)
    object_ = pickle.load(open(path, 'rb'))
    print('Loaded Pickle', path)
    return object_
```

# Load Dataset

In [ ]:

```
cardio = load_dataset()
cardio.columns
```

Out[ ]:

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

In [ ]:

```
cardio.drop('id', axis=1, inplace=True)
save_object(cardio, 'Initial_cardio.dataframe')
cardio.head()
```

```
Saving Pickle E:\Github\SEMESTER-3\PML_Project\Pickles_And_Models\Initial_cardio.datafra
me
```

Out[ ]:

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

# Basic Data Information

In [ ]:
```
cardio.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70000 entries, 0 to 69999
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   age          70000 non-null  int64  
 1   gender       70000 non-null  int64  
 2   height       70000 non-null  int64  
 3   weight       70000 non-null  float64
 4   ap_hi        70000 non-null  int64  
 5   ap_lo        70000 non-null  int64  
 6   cholesterol  70000 non-null  int64  
 7   gluc         70000 non-null  int64  
 8   smoke        70000 non-null  int64  
 9   alco         70000 non-null  int64  
 10  active       70000 non-null  int64  
 11  cardio       70000 non-null  int64  
dtypes: float64(1), int64(11)
memory usage: 6.4 MB
```

In [ ]:
```
cardio.describe(include='all')
```

Out[ ]:

| | age | gender | height | weight | ap_hi | ap_lo | cholesterc |
|---|---|---|---|---|---|---|---|
| **count** | 70000.000000 | 70000.000000 | 70000.000000 | 70000.000000 | 70000.000000 | 70000.000000 | 70000.00000 |
| **mean** | 19468.865814 | 1.349571 | 164.359229 | 74.205690 | 128.817286 | 96.630414 | 1.36687 |
| **std** | 2467.251667 | 0.476838 | 8.210126 | 14.395757 | 154.011419 | 188.472530 | 0.68025 |
| **min** | 10798.000000 | 1.000000 | 55.000000 | 10.000000 | -150.000000 | -70.000000 | 1.00000 |
| **25%** | 17664.000000 | 1.000000 | 159.000000 | 65.000000 | 120.000000 | 80.000000 | 1.00000 |
| **50%** | 19703.000000 | 1.000000 | 165.000000 | 72.000000 | 120.000000 | 80.000000 | 1.00000 |
| **75%** | 21327.000000 | 2.000000 | 170.000000 | 82.000000 | 140.000000 | 90.000000 | 2.00000 |
| **max** | 23713.000000 | 2.000000 | 250.000000 | 200.000000 | 16020.000000 | 11000.000000 | 3.00000 |

# Checking for Null Values

In [ ]:
```
cardio[cardio.isnull().any(axis=1)]
## We can see that there were no null values in dataset, we will move further with Dupl
```

Out[ ]:

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

# Checking for Duplicate Values

In [ ]:
```
cardio_duplicate_values = cardio[cardio.duplicated(keep=False)]
print('Shape of duplicates :',cardio_duplicate_values.shape)
cardio_duplicate_values.sort_values(by=['age']).head()
```

```
Shape of duplicates : (48, 12)
```

Out[ ]:

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

| | age | gender | height | weight | ap_hi | ap_lo | cholesterol | gluc | smoke | alco | active | cardio |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **40365** | 14552 | 1 | 158 | 64.0 | 120 | 80 | 1 | 1 | 0 | 0 | 1 | 0 |
| **6325** | 14552 | 1 | 158 | 64.0 | 120 | 80 | 1 | 1 | 0 | 0 | 1 | 0 |
| **64169** | 16160 | 1 | 168 | 65.0 | 120 | 80 | 1 | 1 | 0 | 0 | 1 | 1 |
| **17101** | 16160 | 1 | 168 | 65.0 | 120 | 80 | 1 | 1 | 0 | 0 | 1 | 1 |
| **1204** | 16793 | 1 | 165 | 68.0 | 120 | 80 | 1 | 1 | 0 | 0 | 1 | 0 |

In [ ]:

```
# We can observe that 24 instances were being duplicated, we will try to remove them fr
cardio.drop(index=cardio[cardio.duplicated()].index, axis=0, inplace=True)
print('After removing duplicates :', cardio.shape)
```

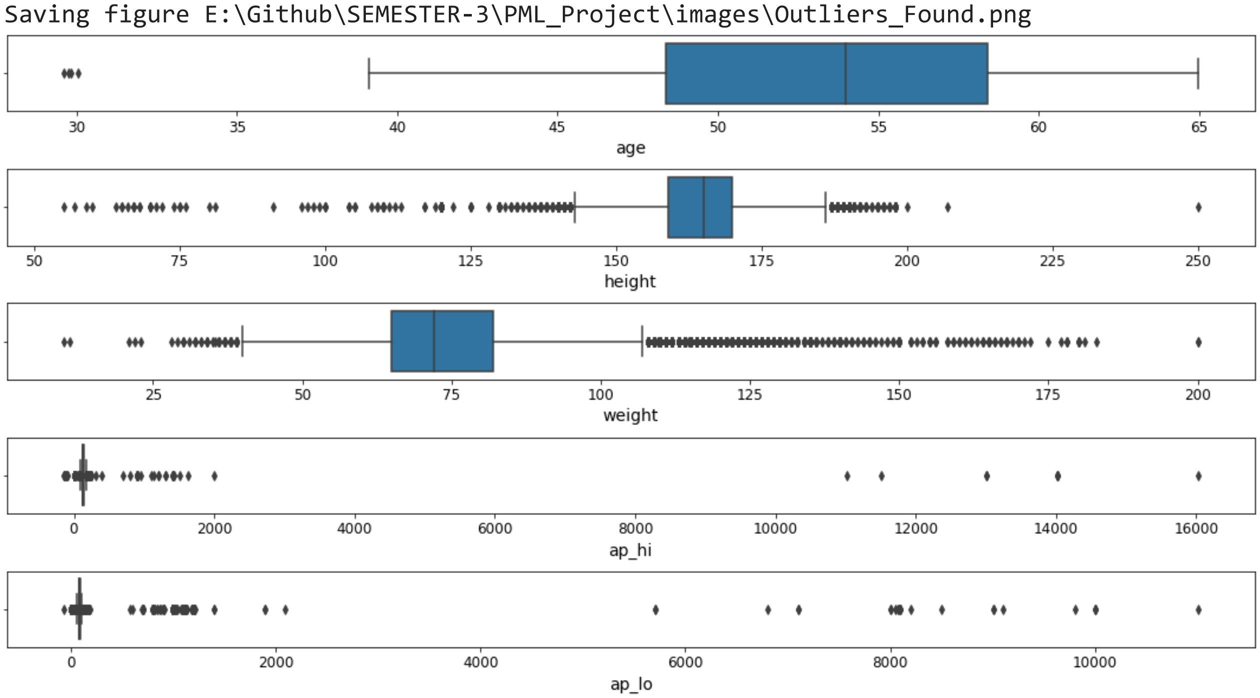
```
After removing duplicates : (69976, 12)
```

# Outliers Detection

In [ ]:

```
# Outliers Detection
continuous_features = ['age', 'height', 'weight', 'ap_hi', 'ap_lo']
fig, axes = plt.subplots(nrows=len(continuous_features), ncols=1, figsize =(15,8));

sns.boxplot(cardio['age']/365, ax=axes[0])

for (index, column) in enumerate(continuous_features[1:], 1):
    sns.boxplot(cardio[column], ax=axes[index])

save_fig('Outliers_Found')
plt.show()
```

```
Saving figure E:\Github\SEMESTER-3\PML_Project\images\Outliers_Found.png
```




In [ ]:

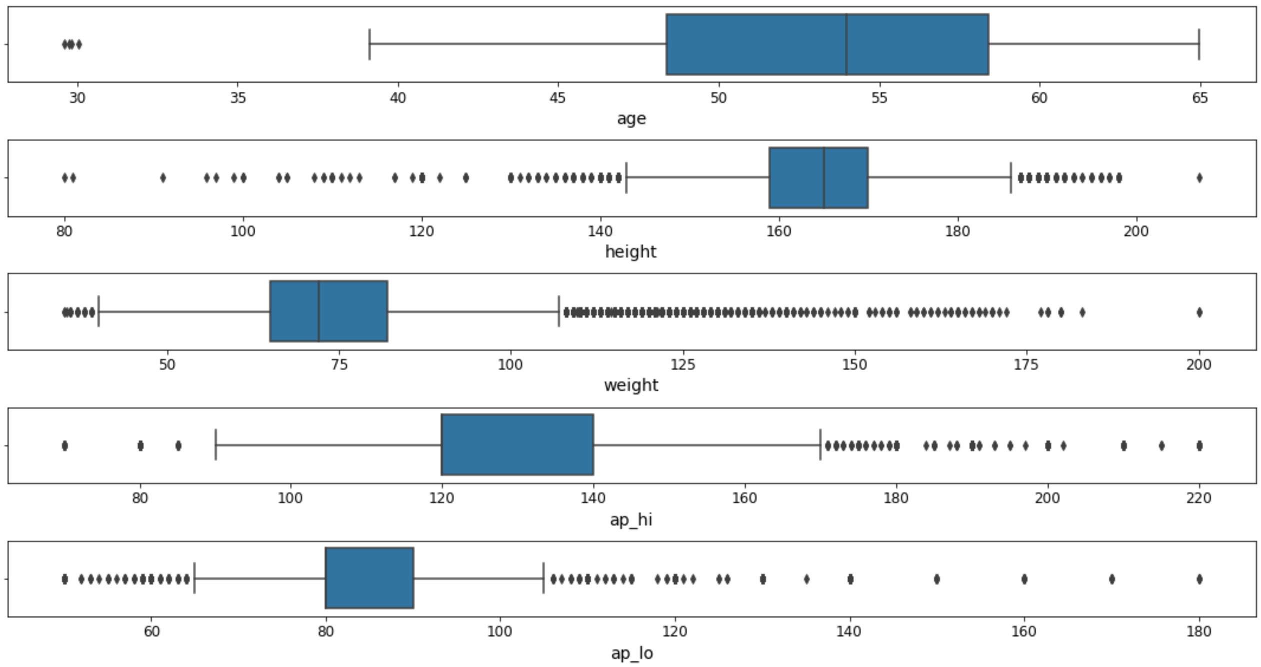
```
# We will do manual adjustment/deletion instead of zscore method due to it being medica

cardio.drop(cardio[(cardio['height'] < 80) | (cardio['height'] > 220)].index, inplace=T
cardio.drop(cardio[cardio['weight'] < 35].index, inplace=True)
```

```
cardio.drop(cardio[(cardio['ap_hi'] < 50) | (cardio['ap_hi'] > 225)].index, inplace=Tru
cardio.drop(cardio[(cardio['ap_lo'] < 50) | (cardio['ap_lo'] > 180)].index, inplace=Tru
```

In [ ]:

```
# Outliers Detection
continuous_features = ['age', 'height', 'weight', 'ap_hi', 'ap_lo']
fig, axes = plt.subplots(nrows=len(continuous_features), ncols=1, figsize =(15,8));

sns.boxplot(cardio['age']/365, ax=axes[0])

for (index, column) in enumerate(continuous_features[1:], 1):
    sns.boxplot(cardio[column], ax=axes[index])

save_fig('Outliers_Removed')
plt.show()
```

```
Saving figure E:\Github\SEMESTER-3\PML_Project\images\Outliers_Removed.png
```



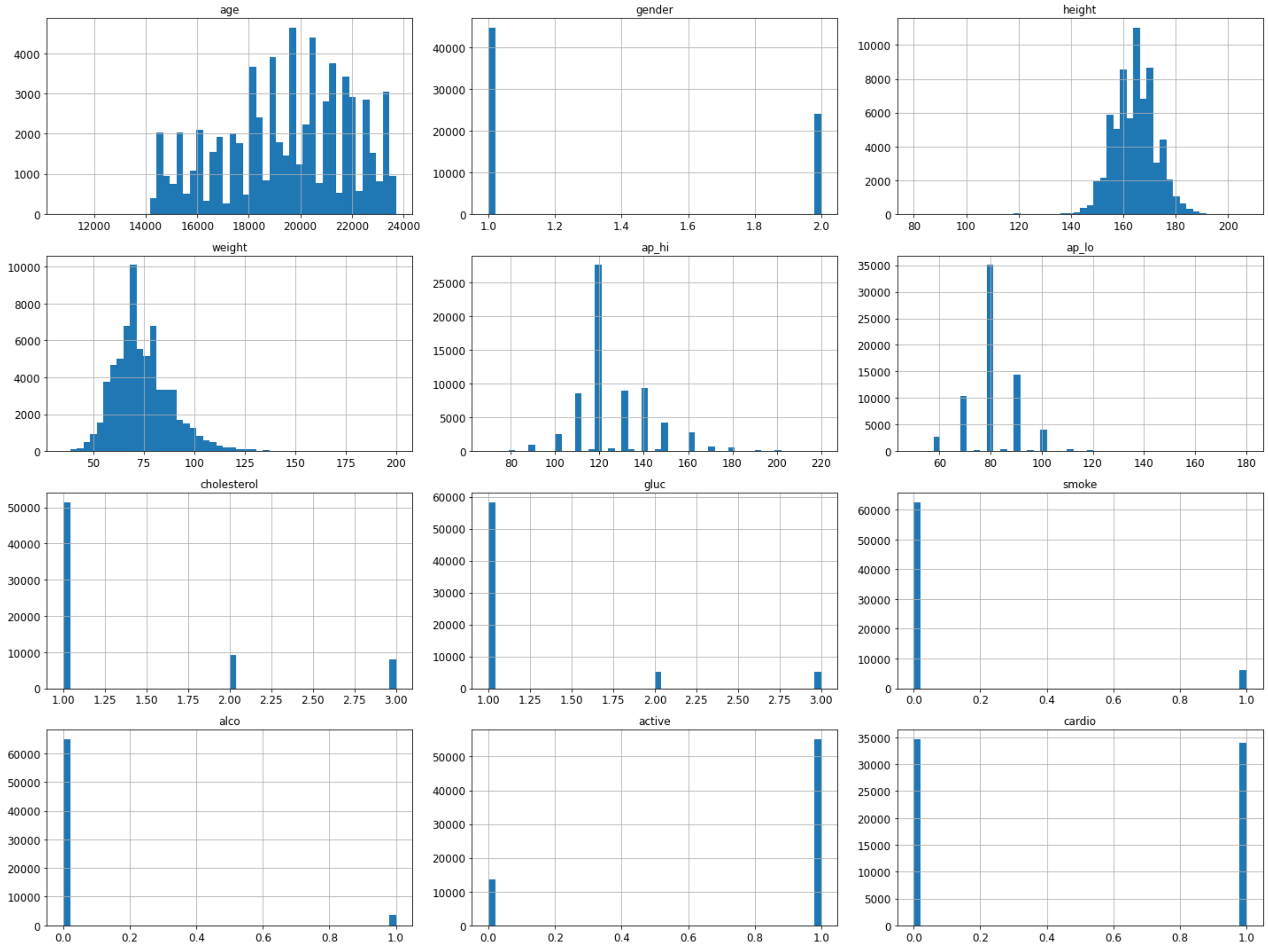

We can observe that after implementing Outlier removal we have got more useful data. Majority of the Outliers presented in **ap_hi**, **ap_lo** and **height** columns.

# Histogram for all features each with 50 bins

In [ ]:

```
cardio.hist(bins=50, figsize=(20,15))
save_fig('Cardio_Histogram_Full')
plt.show()
```

```
Saving figure E:\Github\SEMESTER-3\PML_Project\images\Cardio_Histogram_Full.png
```



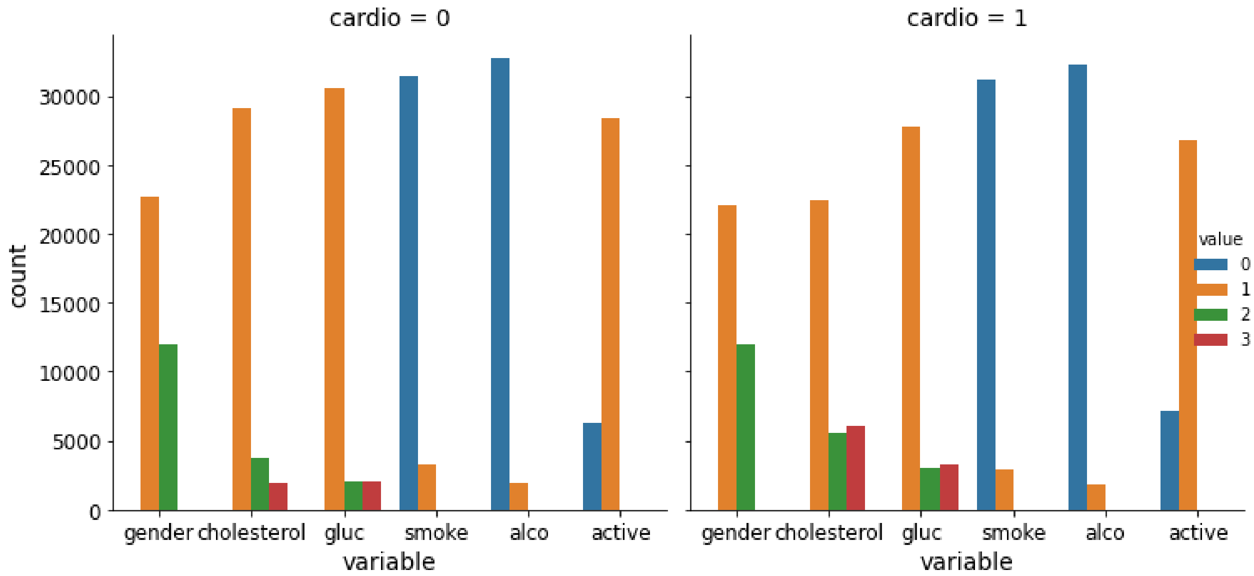

From the above plots, we have observed that **Cholesterol** and **Glucose** have each 3 classes in them. We have to seperate them into different columns.

# Distributions of Categorical Features

In [ ]:

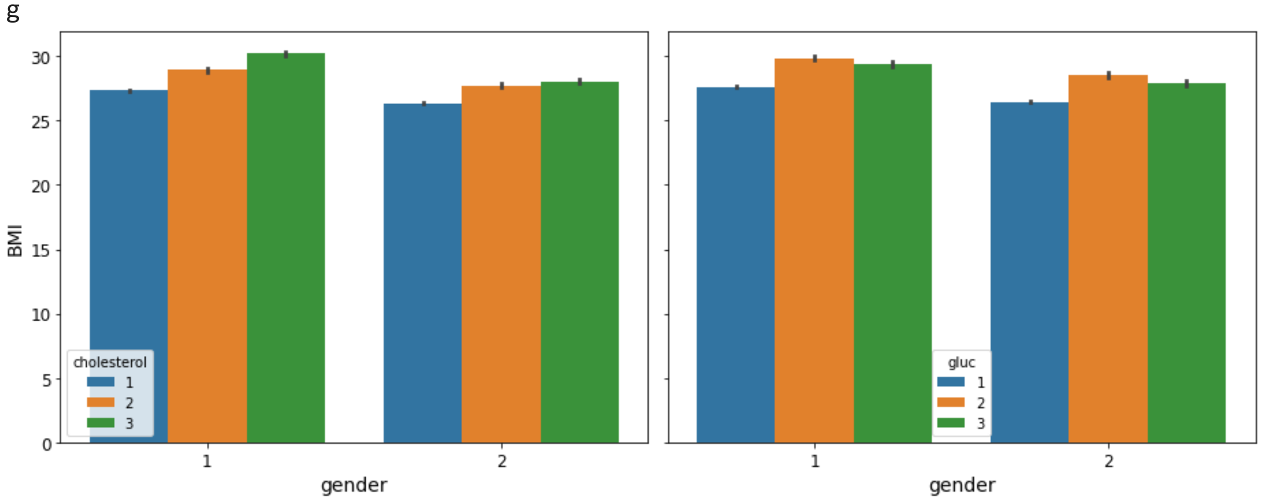
```
cat_features = ['gender','cholesterol','gluc', 'smoke', 'alco', 'active']
df_long = pd.melt(cardio, id_vars=['cardio'], value_vars=cat_features)
sns.catplot(x="variable", hue="value", col="cardio",data=df_long, kind="count")
save_fig('Categorical_Features_Comparison')
plt.show()
```

```
Saving figure E:\Github\SEMESTER-3\PML_Project\images\Categorical_Features_Comparison.pn
g
```




In [ ]:
```
cat_column = ['cholesterol','gluc']

fig, axes = plt.subplots(nrows=1, ncols=len(cat_column), figsize =(13,5), sharey=True)
fig.subplots_adjust(hspace=0.4, wspace=0.4)
BMI = cardio['weight']/((cardio['height']/100)**2)

for (index, column) in enumerate(cat_column):
    sns.barplot(x='gender',y=BMI, data=cardio, hue=column, ax=axes[index])

axes[0].set_ylabel('BMI')
save_fig('BMI_vs_Gender_Category_Analysis')
plt.show()
```

```
Saving figure E:\Github\SEMESTER-3\PML_Project\images\BMI_vs_Gender_Category_Analysis.pn
g
```




# Dataset Splitting

## Splitting of Cardio Dataset into training and testing set with Stratified Shuffle Split method

In [ ]:
```
save_object(cardio, 'Final_cardio.dataframe')
```

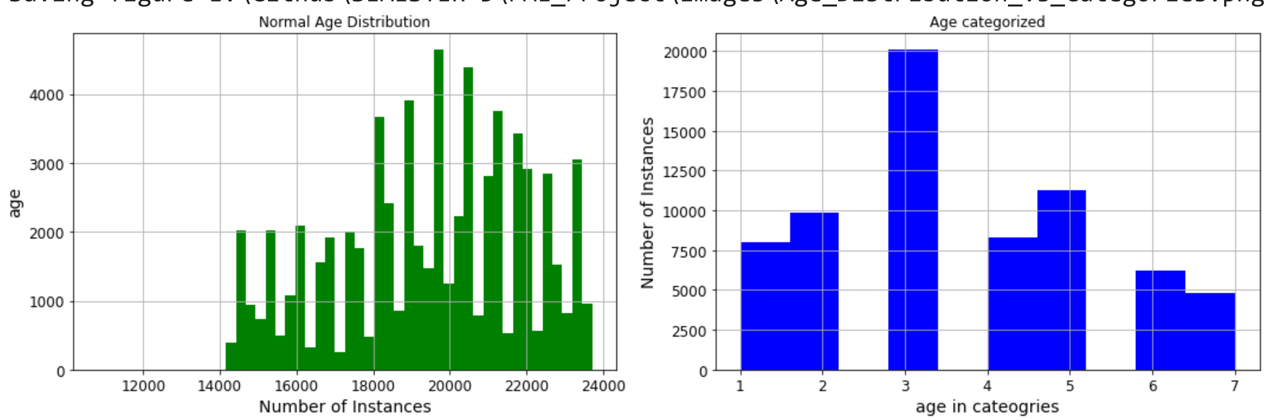
```
Saving Pickle E:\Github\SEMESTER-3\PML_Project\Pickles_And_Models\Final_cardio.dataframe
```

In [ ]:
```
cardio['age_cat'] = pd.cut(cardio['age'],
                           bins=[0, 16000, 18000, 20000, 21000, 22000, 23000, np.inf],
                           labels=[1 , 2, 3, 4, 5, 6, 7])
cardio['age_cat'].value_counts()
```

Out[ ]:
```
3    20111
5    11292
2     9859
4     8319
1     8035
6     6249
7     4816
Name: age_cat, dtype: int64
```

In [ ]:
```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15,5))

cardio['age'].hist(bins=50, ax=axes[0], color='g')
axes[0].set(title= 'Normal Age Distribution', xlabel='Number of Instances', ylabel='age

cardio['age_cat'].hist(ax=axes[1], color='b')
axes[1].set(title= 'Age categorized', xlabel='age in cateogries', ylabel='Number of Ins

save_fig('Age_Distribution_vs_Categories')
plt.show()
```

```
Saving figure E:\Github\SEMESTER-3\PML_Project\images\Age_Distribution_vs_Categories.png
```




cardio_train, cardio_test = train_test_split(cardio, test_size=0.2)

fig, split = plt.subplots(nrows=1, ncols=2, figsize=(15,5))

cardio_train['age_cat'].hist(ax=split[0], color='g') split[0].set(title= 'Default Train Set', xlabel='age in bins', ylabel='Number of Instances')

cardio_test['age_cat'].hist(ax=split[1], color='b') split[1].set(title= 'Default Test Set', xlabel='age in bins', ylabel='Number of Instances')

plt.tight_layout() plt.show()

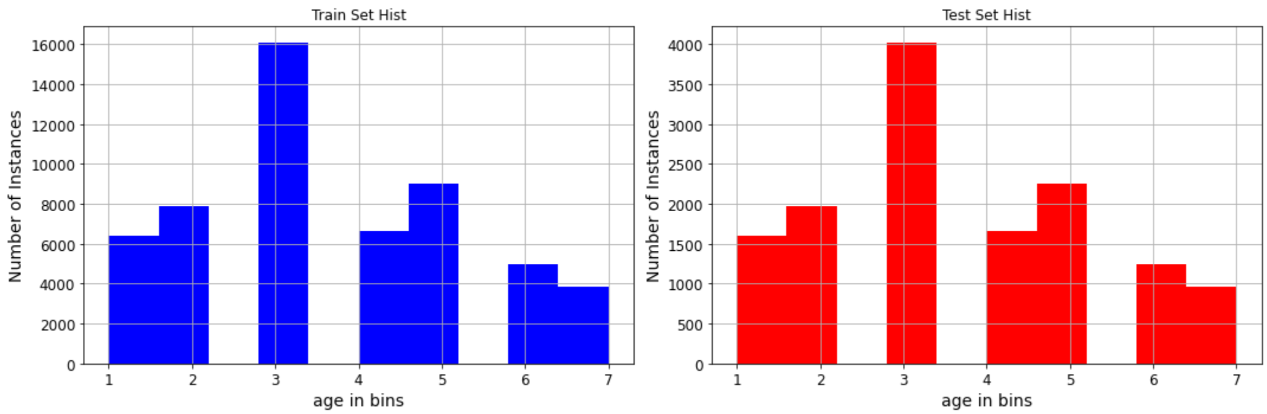
save_object(cardio, 'qwicksave.data')

In [ ]:
```
strat_split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in strat_split.split(cardio, cardio['age_cat']):
    strat_train_set = cardio.iloc[train_index]
    strat_test_set  = cardio.iloc[test_index]
```

In [ ]:
```
fig, strat = plt.subplots(nrows=1, ncols=2, figsize=(15,5))
```

```
strat_train_set['age_cat'].hist(ax=strat[0], color='b')
strat[0].set(title= 'Train Set Hist', xlabel='age in bins', ylabel='Number of Instances

strat_test_set['age_cat'].hist(ax=strat[1], color='r')
strat[1].set(title= 'Test Set Hist', xlabel='age in bins', ylabel='Number of Instances'

save_fig('Stratified_Train_Test_Set')
plt.show()
```

```
Saving figure E:\Github\SEMESTER-3\PML_Project\images\Stratified_Train_Test_Set.png
```




In [ ]:
```
for set_ in (strat_train_set, strat_test_set):
    set_.drop('age_cat', axis=1, inplace=True)
```

In [ ]:
```
cardio = strat_train_set.copy()
```

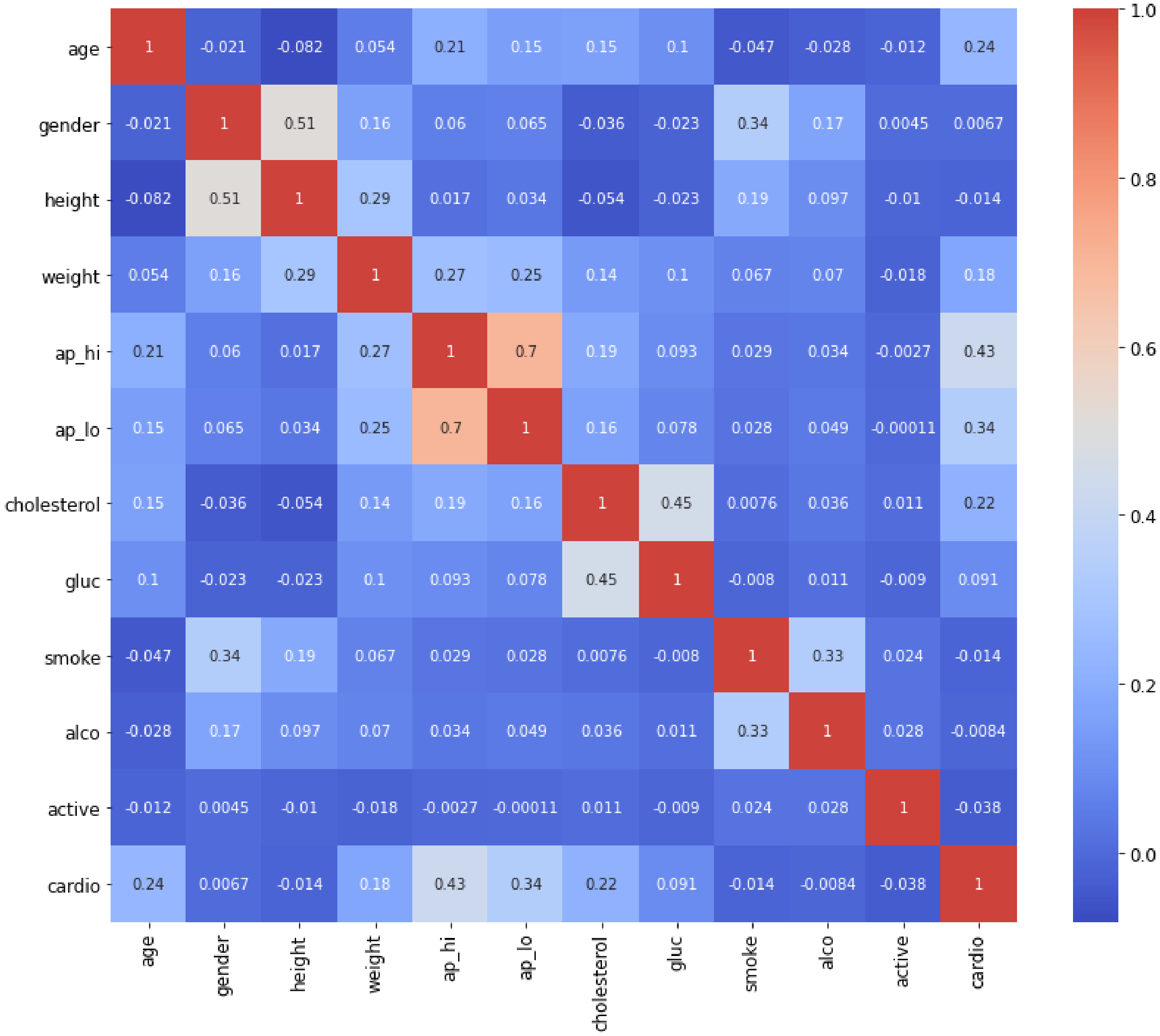
In [ ]:
```
corr_matrix = cardio.corr()
```

In [ ]:
```
corr_matrix['cardio'].sort_values(ascending=False)
```

Out[ ]:
```
cardio         1.000000
ap_hi          0.427615
ap_lo          0.339710
age            0.239476
cholesterol    0.219276
weight         0.176428
gluc           0.091428
gender         0.006667
alco          -0.008415
height        -0.013512
smoke         -0.014269
active        -0.038482
Name: cardio, dtype: float64
```

In [ ]:
```
# Correlation HeatMap
fig, axes = plt.subplots(1,1,figsize=(12,10))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', ax=axes, center=0.5)
save_fig('Heat_Map_General')
plt.show()
```

```
Saving figure E:\Github\SEMESTER-3\PML_Project\images\Heat_Map_General.png
```




As we can see that the data is not very much correlated, so we proceed further with Data Tranformation Pipelines

In [ ]:
```
# save_object(strat_train_set.drop(columns='cardio',axis=1), 'strat_train_cardio.datafr
# save_object(strat_train_set['cardio'], 'strat_train_cardio_labels.dataframe')
# save_object(strat_test_set.drop(columns='cardio',axis=1), 'strat_test_cardio.datafram
# save_object(strat_test_set['cardio'], 'strat_test_cardio_labels.dataframe')

# cardio = strat_train_set.drop(columns='cardio',axis=1).copy()
# cardio_labels = strat_train_set['cardio'].copy()
```

# Data Transformation Pipelines

In [4]:
```
cardio = load_object('Final_cardio.dataframe')

X_train = load_object('strat_train_cardio.dataframe')
y_train = load_object('strat_train_cardio_labels.dataframe')

X_test = load_object('strat_test_cardio.dataframe')
y_test = load_object('strat_test_cardio_labels.dataframe')
```

```
Loaded Pickle /content/Pickles_And_Models/Final_cardio.dataframe
```

```
Loaded Pickle /content/Pickles_And_Models/strat_train_cardio.dataframe
Loaded Pickle /content/Pickles_And_Models/strat_train_cardio_labels.dataframe
Loaded Pickle /content/Pickles_And_Models/strat_test_cardio.dataframe
Loaded Pickle /content/Pickles_And_Models/strat_test_cardio_labels.dataframe
```

In [5]:

```
class DataFrameSelector(BaseEstimator, TransformerMixin):

    def __init__(self, attribute_names):
        self.attribute_names = attribute_names

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        return X[self.attribute_names].values

target_feature = ['cardio']
cat_attributes = ['cholesterol', 'gluc']
num_attributes = ['age', 'gender', 'height', 'weight', 'ap_hi', 'ap_lo', 'smoke', 'alco
new_cat_attributes = ['cholesterol_1', 'cholesterol_2', 'cholesterol_3', 'gluc_1', 'glu

num_pipeline = Pipeline([
    ('selector'  , DataFrameSelector(num_attributes)),
    ('imputer'   , SimpleImputer(strategy='median')),
    ('std_scaler', StandardScaler())
])

cat_pipeline = Pipeline([
    ('selector'    , DataFrameSelector(cat_attributes)),
    ('cat_encoder' , OneHotEncoder(sparse=False))
])

full_pipeline = FeatureUnion(transformer_list=[
    ('Numerical_Pipeline', num_pipeline),
    ('Categorical_Pipeline', cat_pipeline)
])
```

In [6]:

```
X_train.head(2)
```

Out[6]:

| | age | gender | height | weight | ap_hi | ap_lo | cholesterol | gluc | smoke | alco | active |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **3280** | 22640 | 2 | 167 | 78.0 | 160 | 100 | 1 | 1 | 0 | 0 | 1 |
| **40230** | 20613 | 1 | 156 | 88.0 | 150 | 80 | 1 | 1 | 0 | 0 | 1 |

In [7]:

```
full_pipeline.fit(X_train)
```

Out[7]:

```
FeatureUnion(n_jobs=None,
             transformer_list=[('Numerical_Pipeline',
                                Pipeline(memory=None,
                                         steps=[('selector',
                                                 DataFrameSelector(attribute_names=['ag
e',
                                                                                    'gen
der',
                                                                                    'hei
ght',
                                                                                    'wei
ght',
                                                                                    'ap_
hi',
```

```
                                                                                          'ap_
lo',
                                                                                          'smo
ke',
                                                                                          'alc
o',
                                                                                          'act
ive'])),
                                                ('imputer',
                                                 SimpleImputer(add_indicator=False,
                                                               copy=True,
                                                               fill_value=None,
                                                               missing_values=nan,
                                                               strategy='median',
                                                               verbose=0)),
                                                ('std_scaler',...
                                                                with_mean=True,
                                                                with_std=True))],
                                         verbose=False)),
                                ('Categorical_Pipeline',
                                 Pipeline(memory=None,
                                          steps=[('selector',
                                                  DataFrameSelector(attribute_names=['cho
lesterol',
                                                                                          'glu
c'])),
                                                ('cat_encoder',
                                                 OneHotEncoder(categories='auto',
                                                               drop=None,
                                                               dtype=<class 'numpy.float
64'>,

                                                               handle_unknown='error',
                                                               sparse=False))],
                                         verbose=False))],
             transformer_weights=None, verbose=False)
```

In [8]:
```
save_object(full_pipeline, 'full_pipeline.transformer')
```

```
Saving Pickle /content/Pickles_And_Models/full_pipeline.transformer
```

In [9]:
```
X_train = pd.DataFrame(data=full_pipeline.transform(X_train), columns=num_attributes+ne
X_test = pd.DataFrame(data=full_pipeline.transform(X_test), columns=num_attributes+new_
```

In [10]:
```
X_train.head(2)
```

Out[10]:

| | age | gender | height | weight | ap_hi | ap_lo | smoke | alco | active | choleste |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1.286661 | 1.371339 | 0.325114 | 0.266796 | 1.997143 | 1.941350 | -0.310522 | -0.237641 | 0.494967 | |
| **1** | 0.465335 | -0.729214 | -1.048238 | 0.965196 | 1.398966 | -0.143682 | -0.310522 | -0.237641 | 0.494967 | |

# Machine Learning Algorithms Implementation

## Logistic Regression

In [ ]:
```
grid_values={
    "C" : [0.001,0.01,1,5,10,25,50],
    "penalty" : ["l1","l2"]
```

```
}

grid_search=GridSearchCV(LogisticRegression(),grid_values,cv=10)
grid_search.fit(X_train,y_train)

print("tuned hpyerparameters :(best parameters) ",grid_search.best_params_)
print("accuracy :",grid_search.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 25, 'penalty': 'l2'}
accuracy : 0.7276502482814405
```

In [ ]:

```
#Training the model with the optimised parameters
log_reg = LogisticRegression(C = 25, penalty = 'l2')
log_reg.fit(X_train, y_train)

#KFold cross-validation for evaluating the model
cv = KFold(n_splits=10, random_state=1, shuffle=True)
scores = cross_val_score(log_reg, X_train, y_train, scoring='recall', cv=cv, n_jobs=-1)

# Reporting the performance
print('Accuracy: %.3f | Scores: (%.3f)' % (np.mean(scores), np.std(scores)))
```

```
Accuracy: 0.669 | Scores: (0.009)
```

In [ ]:

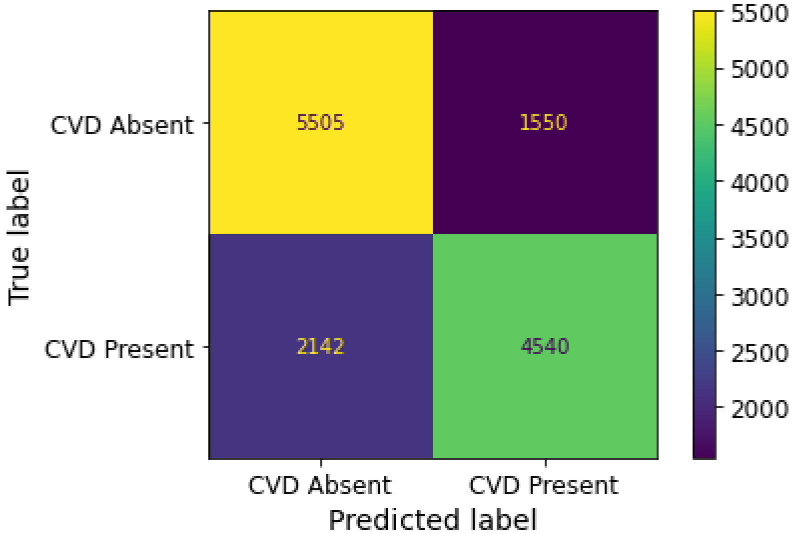
```
# Testing Set
log_reg = LogisticRegression(C = 25, penalty = 'l2')
log_reg.fit(X_train, y_train)
log_pred_test = log_reg.predict(X_test)
print('Testing dataset')
print(confusion_matrix(y_test,log_pred_test))
print(classification_report(y_test,log_pred_test))
print(accuracy_score(y_test,log_pred_test))
```

```
Testing dataset
[[5505 1550]
 [2142 4540]]
              precision    recall  f1-score   support

           0       0.72      0.78      0.75      7055
           1       0.75      0.68      0.71      6682

    accuracy                           0.73     13737
   macro avg       0.73      0.73      0.73     13737
weighted avg       0.73      0.73      0.73     13737

0.731236805707214
```

In [ ]:

```
plot_confusion_matrix(log_reg, X_test, y_test, display_labels = ['CVD Absent', 'CVD Pre
save_fig('Confusion_matrix_Logisitic')
plt.show()
```

```
Saving figure E:\Github\SEMESTER-3\PML_Project\images\Confusion_matrix_Logisitic.png
```




In [ ]:
```
save_object(log_reg, 'Logistic_Regression.model')
```

```
Saving Pickle E:\Github\SEMESTER-3\PML_Project\Pickles_And_Models\Logistic_Regression.mo
del
```

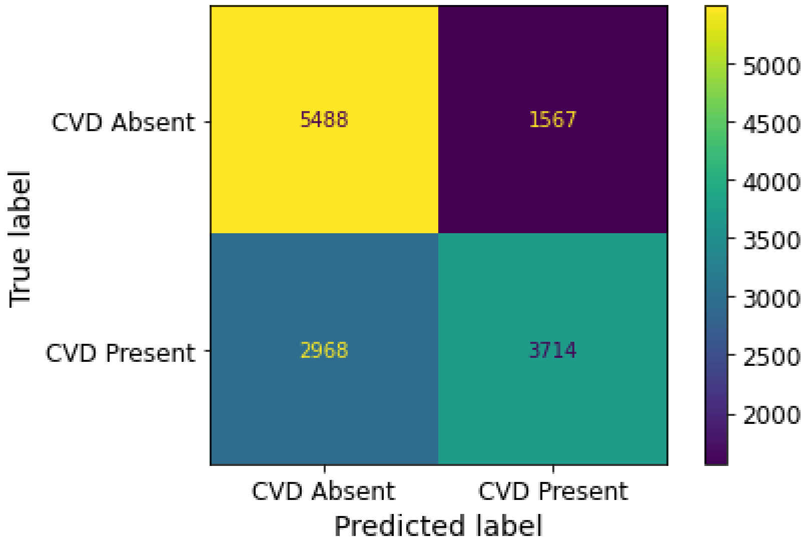
# Naive Bayes

In [ ]:
```
gauss_nb = GaussianNB()
gauss_nb.fit(X_train, y_train)
gnb_pred_test = gauss_nb.predict(X_test)

print('Testing dataset')
print(confusion_matrix(y_test,gnb_pred_test))
print(classification_report(y_test,gnb_pred_test))
print(accuracy_score(y_test,gnb_pred_test))
```

```
Testing dataset
[[5488 1567]
 [2968 3714]]
              precision    recall  f1-score   support

           0       0.65      0.78      0.71      7055
           1       0.70      0.56      0.62      6682

    accuracy                           0.67     13737
   macro avg       0.68      0.67      0.66     13737
weighted avg       0.68      0.67      0.67     13737

0.6698696949843488
```

In [ ]:
```
plot_confusion_matrix(gauss_nb, X_test, y_test, display_labels = ['CVD Absent', 'CVD Pr
save_fig('Confusion_matrix_Naive_Bayes')
plt.show()
```

```
Saving figure E:\Github\SEMESTER-3\PML_Project\images\Confusion_matrix_Naive_Bayes.png
```




In [ ]:
```
save_object(gauss_nb, 'Naive_Bayes.model')
```

```
Saving Pickle E:\Github\SEMESTER-3\PML_Project\Pickles_And_Models\Naive_Bayes.model
```

# K Nearest Neighbors

In [ ]:
```
hyperparameters={
    "n_neighbors" : [1, 5, 10, 25, 50, 75],
    "p" : [1, 2] # manhattan_distance (l1), and euclidean_distance (l2)
}

grid_search=GridSearchCV(KNeighborsClassifier(),hyperparameters,cv=10, verbose=1, n_job
grid_search.fit(X_train,y_train)

print("tuned hpyerparameters :(best parameters) ",grid_search.best_params_)
print("accuracy :",grid_search.best_score_)
```

```
Fitting 10 folds for each of 12 candidates, totalling 120 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done 120 out of 120 | elapsed:  4.6min finished
tuned hpyerparameters :(best parameters)  {'n_neighbors': 75, 'p': 1}
accuracy : 0.7289059087703583
```

In [ ]:
```
#Training the model with the optimised parameters
knn = KNeighborsClassifier(n_neighbors = 75, p = 1)
knn.fit(X_train, y_train)

#KFold cross-validation for evaluating the model
cv = KFold(n_splits=10, random_state=1, shuffle=True)
scores = cross_val_score(knn, X_train, y_train, scoring='accuracy', cv=cv, n_jobs=-1)

# Reporting the performance
print('Scores -> Mean: %.3f | STD: (%.3f)' % (np.mean(scores), np.std(scores)))
```

```
Scores -> Mean: 0.728 | STD: (0.005)
```

In [ ]:
```
# Testing Set
knn = KNeighborsClassifier(n_neighbors = 25, p = 1)
knn.fit(X_train, y_train)
knn_pred_test = knn.predict(X_test)
```

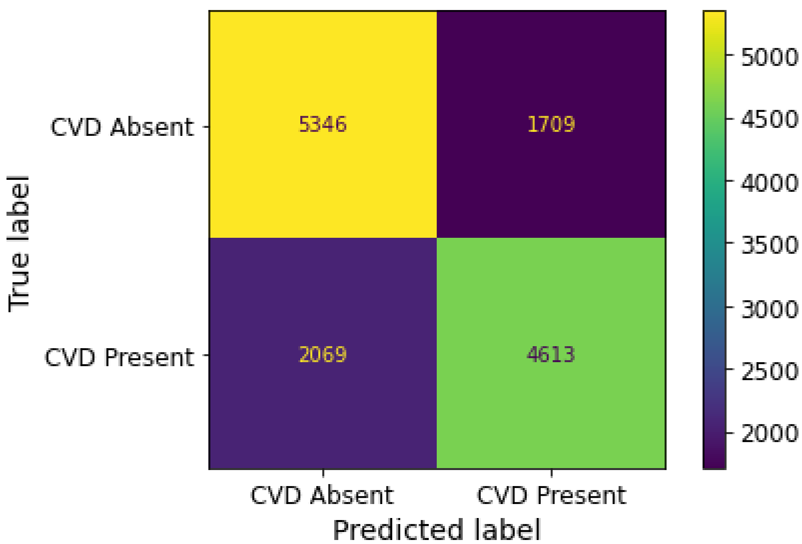
```
print('Testing dataset')
print(confusion_matrix(y_test,knn_pred_test))
print(classification_report(y_test,knn_pred_test))
print(accuracy_score(y_test,knn_pred_test))
```

```
Testing dataset
[[5346 1709]
 [2069 4613]]
              precision    recall  f1-score   support

           0       0.72      0.76      0.74      7055
           1       0.73      0.69      0.71      6682

    accuracy                           0.72     13737
   macro avg       0.73      0.72      0.72     13737
weighted avg       0.73      0.72      0.72     13737

0.7249763412681081
```

In [ ]:

```
plot_confusion_matrix(knn, X_test, y_test, display_labels = ['CVD Absent', 'CVD Present
save_fig('Confusion_Matrix_KNN')
plt.show()
```

```
Saving figure E:\Github\SEMESTER-3\PML_Project\images\Confusion_Matrix_KNN.png
```




In [ ]:

```
save_object(knn, 'KNN.model')
```

```
Saving Pickle E:\Github\SEMESTER-3\PML_Project\Pickles_And_Models\KNN.model
```

# Support Vector Machines (SVM)

In [11]:

```
'''
hyperparameters={
    "C" : [0.1, 10],
    "gamma" : [0.1, 1],
    "kernel" : ['linear']
}

grid_search=GridSearchCV(SVC(), hyperparameters,cv=2, verbose=1, n_jobs=-1)
grid_search.fit(X_train,y_train)

print("tuned hpyerparameters :(best parameters) ",grid_search.best_params_)
print("accuracy :",grid_search.best_score_)
```

```
'''
```

Out[11]:
```
'\nhyperparameters={\n    "C" : [0.1, 10],\n    "gamma" : [0.1, 1],\n    "kernel" : [\'l
inear\']\n}\n\ngrid_search=GridSearchCV(SVC(), hyperparameters,cv=2, verbose=1, n_jobs=-
1)\ngrid_search.fit(X_train,y_train)\n\nprint("tuned hpyerparameters :(best parameters)
",grid_search.best_params_)\nprint("accuracy :",grid_search.best_score_)\n\n'
```

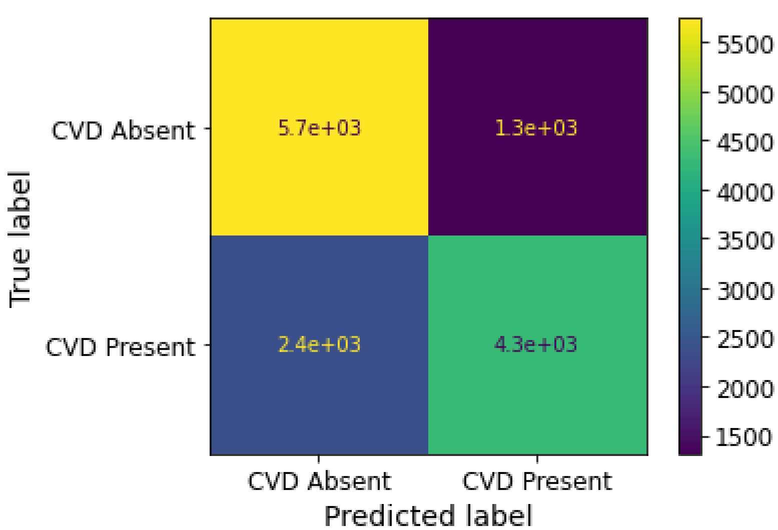
In [ ]:
```
#Training the model with the optimised parameters
svm = SVC(C=0.1, gamma=1, kernel='linear')
svm.fit(X_train, y_train)

#KFold cross-validation for evaluating the model
cv = KFold(n_splits=5, random_state=1, shuffle=True)
scores = cross_val_score(svm, X_train, y_train, scoring='accuracy', cv=cv, n_jobs=-1)

# Reporting the performance
print('Scores -> Mean: %.3f | STD: (%.3f)' % (np.mean(scores), np.std(scores)))
```

In [27]:
```
svm = SVC(C=0.1, gamma=1, kernel='linear')
svm.fit(X_train, y_train)
```

In [24]:
```
svm_pred_test = svm.predict(X_test)
print('Testing dataset')
print(confusion_matrix(y_test,svm_pred_test))
print(classification_report(y_test,svm_pred_test))
print(accuracy_score(y_test,svm_pred_test))
```

```
Testing dataset
[[5744 1311]
 [2387 4295]]
              precision    recall  f1-score   support

           0       0.71      0.81      0.76      7055
           1       0.77      0.64      0.70      6682

    accuracy                           0.73     13737
   macro avg       0.74      0.73      0.73     13737
weighted avg       0.74      0.73      0.73     13737

0.7308000291184392
```

In [26]:
```
plot_confusion_matrix(svm, X_test, y_test, display_labels = ['CVD Absent', 'CVD Present
plt.show()
```



In [25]:
```
save_object(svm, 'SVM.model')
```

```
Saving Pickle /content/Pickles_And_Models/SVM.model
```

# Decision Trees

In [22]:
```
#Training the model with the default parameters
tree_clf = DecisionTreeClassifier()
tree_clf.fit(X_train,y_train)

#KFold cross-validation for evaluating the model
cv = KFold(n_splits=10, random_state=1, shuffle=True)
scores = cross_val_score(tree_clf, X_train,y_train, scoring='accuracy', cv=cv, n_jobs=-

# Reporting the performance
print('Accuracy: %.3f (%.3f)' % (np.mean(scores), np.std(scores)))
```

```
Accuracy: 0.633 (0.005)
```

In [ ]:
```
pipeline = Pipeline([('clf',tree_clf)])
parameters = {
    'clf__max_depth': np.linspace(1,10,10),
    'clf__min_samples_split': (1,2,3,4,5),
    'clf__min_samples_leaf': (1,2,3,4,5)
}
grid_search = GridSearchCV(pipeline, parameters, n_jobs=-1)
grid_search.fit(X_train,y_train)

grid_search.best_score_
```

In [ ]:
```
best_parameters = grid_search.best_estimator_.get_params()
for param_name in sorted(parameters.keys()):
    print(f'{param_name} : {best_parameters[param_name]}')
```

In [ ]:
```
#Training the model with the optimised parameters
tree_clf = DecisionTreeClassifier(max_depth=5,min_samples_leaf=1,min_samples_split=2)
tree_clf.fit(X_train,y_train)

#KFold cross-validation for evaluating the model
cv = KFold(n_splits=10, random_state=1, shuffle=True)
scores = cross_val_score(tree_clf, X_train,y_train, scoring='accuracy', cv=cv, n_jobs=-

# Reporting the performance
print('Accuracy: %.3f (%.3f)' % (np.mean(scores), np.std(scores)))
```

In [23]:
```
save_object(tree_clf, 'Decision_Tree.model')
```

```
Saving Pickle /content/Pickles_And_Models/Decision_Tree.model
```

# Ensemble Methods

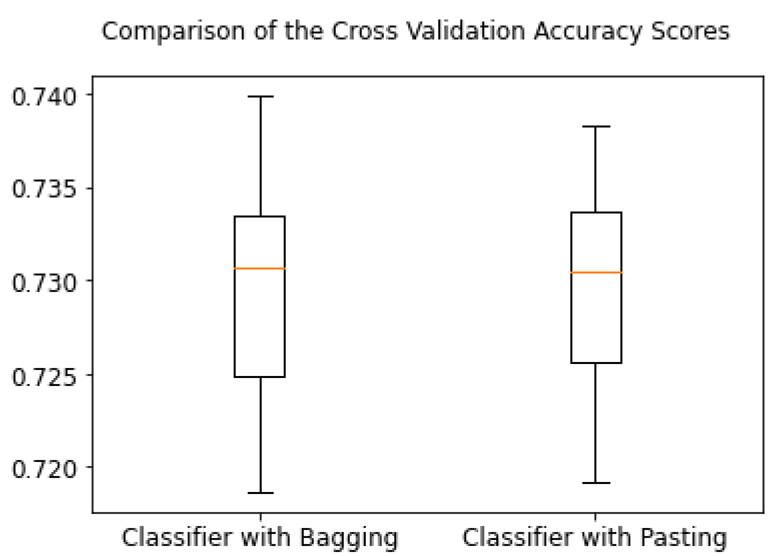
In [16]:
```
def plot_crossval_boxplot(models):
    outcome = []
    model_names = []

    for model_name, model in models:
```

```
        k_fold_validation = KFold(n_splits=10)
        results = cross_val_score(model, X_train, y_train, cv=k_fold_validation, scorin
        outcome.append(results)
        model_names.append(model_name)

    fig = plt.figure()
    fig.suptitle('Comparison of the Cross Validation Accuracy Scores')
    ax = fig.add_subplot(111)
    plt.boxplot(outcome)
    ax.set_xticklabels(model_names)
    plt.show()
```

# Bagging and Pasting Ensemble Methods

In [17]:

```
#Bagging
bag_clf = BaggingClassifier(DecisionTreeClassifier(),
                            n_estimators=500,
                            max_samples=100,
                            bootstrap=True,
                            n_jobs=-1)

#Pasting
pas_clf = BaggingClassifier(DecisionTreeClassifier(),
                            n_estimators=500,
                            max_samples=100,
                            bootstrap=False,
                            n_jobs=-1)

bag_clf.fit(X_train, y_train)
pas_clf.fit(X_train, y_train)

plot_crossval_boxplot([('Classifier with Bagging',bag_clf), ('Classifier with Pasting',
```

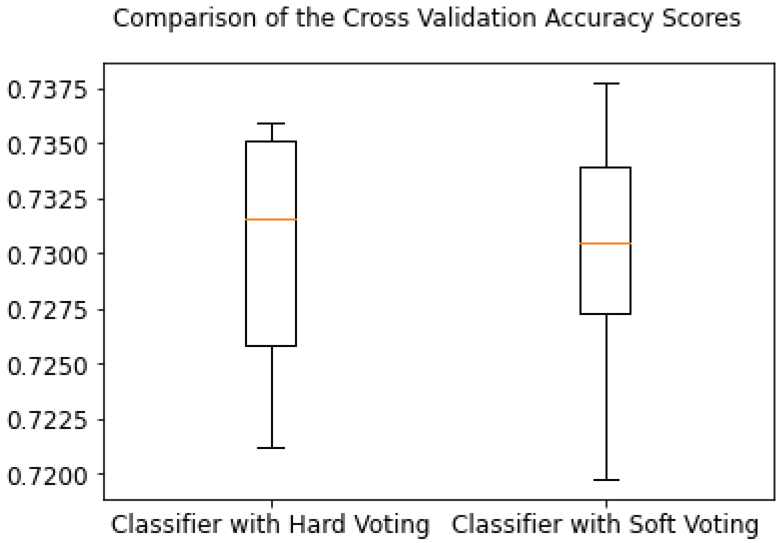



# Hard and Soft Voting in the Ensemble Method

In [18]:

```
#Warning - takes 3 mins to compile
hard_voting_clf = VotingClassifier(estimators=[('bagging', bag_clf), ('pasting', pas_cl
soft_voting_clf = VotingClassifier(estimators=[('bagging', bag_clf), ('pasting', pas_cl
```

```
hard_voting_clf.fit(X_train, y_train)
soft_voting_clf.fit(X_train, y_train)

plot_crossval_boxplot([('Classifier with Hard Voting',hard_voting_clf), ('Classifier wi
```



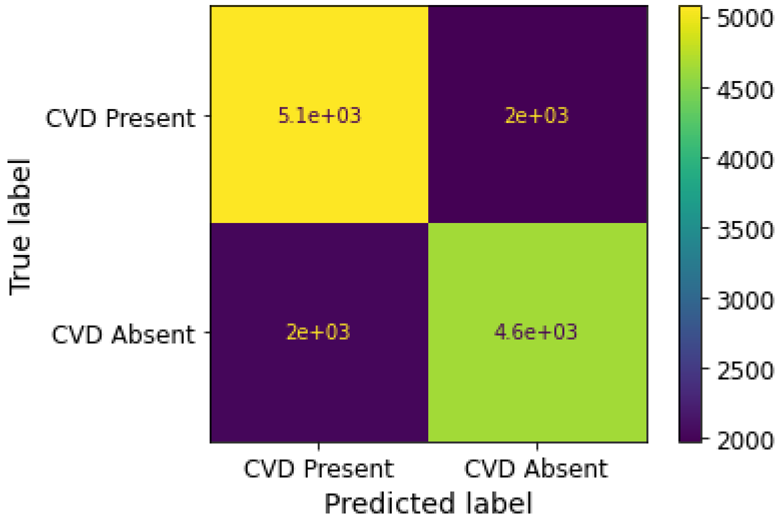

# Random Forest Classifier (Similar to Bagging with Decision Tree Algorithm)

In [31]:
```
# Training the model with the default parameters
rf_clf = RandomForestClassifier()
rf_clf.fit(X_train, y_train)

# KFold cross-validation for evaluating the model
cv = KFold(n_splits=10, random_state=1, shuffle=True)
scores = cross_val_score(rf_clf, X_test, y_test, scoring='accuracy', cv=cv, n_jobs=-1)

# Reporting the performance
print('Accuracy: %.3f (%.3f)' % (np.mean(scores), np.std(scores)))
```

```
Accuracy: 0.717 (0.012)
```

In [32]:
```
#How well can the default paramters predict?
y_pred = rf_clf.predict(X_test)
plot_confusion_matrix(rf_clf, X_test, y_test, display_labels = ['CVD Present','CVD Abse
```

Out[32]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f0eec0bb588>
```




In [33]:

```
# Warning takes 5 mins to compile
'''
pipeline = Pipeline([('clf',rf_clf)])
parameters = {
    'clf__max_depth': (1,2,3,4,5),
    'clf__min_samples_split': (1,2,3),
    'clf__min_samples_leaf': (1,2,3)
}
grid_search = GridSearchCV(pipeline, parameters, n_jobs=-1)
grid_search.fit(X_train,y_train)

grid_search.best_score_
Best score obtianed = 0.7297333333333335
'''
```

Out[33]:

```
"\npipeline = Pipeline([('clf',rf_clf)])\nparameters = {\n    'clf__max_depth': (1,2,3,
4,5),\n    'clf__min_samples_split': (1,2,3),\n    'clf__min_samples_leaf': (1,2,3)\n}\n
grid_search = GridSearchCV(pipeline, parameters, n_jobs=-1)\ngrid_search.fit(X_train,y_t
rain)\n\ngrid_search.best_score_\nBest score obtianed = 0.7297333333333335\n"
```

In [34]:

```
best_parameters = grid_search.best_estimator_.get_params()
for param_name in sorted(parameters.keys()):
  print(f'{param_name} : {best_parameters[param_name]}')

'''
Best Estimators obtained:
clf__max_depth : 5
clf__min_samples_leaf : 1
clf__min_samples_split : 2
'''
```

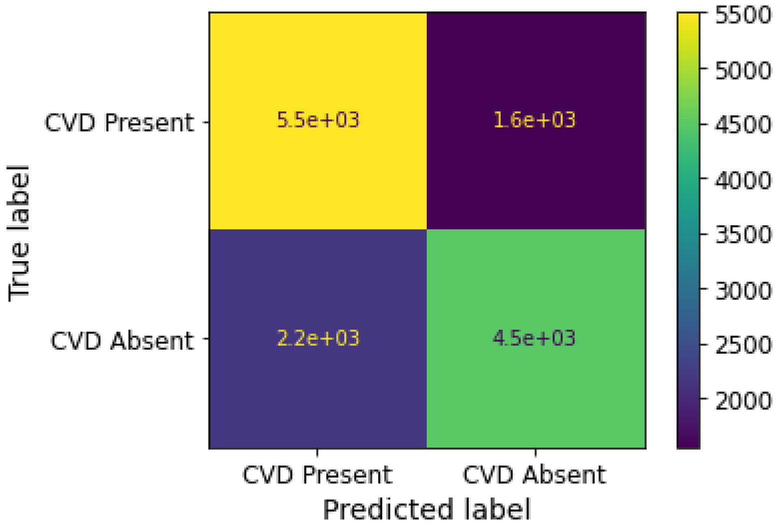
In [35]:

```
#Training the model with the optimised parameters
rf_clf1 = RandomForestClassifier(max_depth=5,min_samples_leaf=1,min_samples_split=2)
rf_clf1.fit(X_train, y_train)

#KFold cross-validation for evaluating the model
cv = KFold(n_splits=10, random_state=1, shuffle=True)
scores = cross_val_score(rf_clf1, X_test, y_test, scoring='accuracy', cv=cv, n_jobs=-1)

# Reporting the performance
print('Accuracy: %.3f (%.3f)' % (np.mean(scores), np.std(scores)))
```

```
Accuracy: 0.728 (0.015)
```

In [36]:
```
#How well can the default paramters predict?
y_pred = rf_clf1.predict(X_test)
plot_confusion_matrix(rf_clf1, X_test, y_test, display_labels = ['CVD Present','CVD Abs
```

Out[36]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f0eebfe3cc0>




In [37]:
```
save_object(rf_clf1, 'Random_Forest.model')
```

```
Saving Pickle /content/Pickles_And_Models/Random_Forest.model
```

# Model Analysis

In [42]:
```
log_reg = load_object('Logistic_Regression.model')
gnb = load_object('Naive_Bayes.model')
knn = load_object('KNN.model')
svm = load_object('SVM.model')
decision_tree = load_object('Decision_Tree.model')
random_forest = load_object('Random_Forest.model')
```

```
Loaded Pickle /content/Pickles_And_Models/Logistic_Regression.model
Loaded Pickle /content/Pickles_And_Models/Naive_Bayes.model
Loaded Pickle /content/Pickles_And_Models/KNN.model
Loaded Pickle /content/Pickles_And_Models/SVM.model
Loaded Pickle /content/Pickles_And_Models/Decision_Tree.model
Loaded Pickle /content/Pickles_And_Models/Random_Forest.model
```

In [65]:
```
models      = [log_reg, gnb, knn, svm, decision_tree, random_forest]
models_cols = ['Logistic_Regression', 'Naive_Bayes', 'KNN', 'Support_Vector_Machines',
```

In [66]:
```
predictions = [model.predict(X_test) for model in models]
conf_mat =[confusion_matrix(y_test, model_pred).reshape(1,-1).tolist()[0] for model_pre
accuracy_score_table = [accuracy_score(y_test, model_pred) for model_pred in prediction
```

In [70]:
```
final_comparison = pd.DataFrame( data=conf_mat, index=models_cols , columns= ['True CVD
```

In [72]:
```
final_comparison['accuracy_score'] = accuracy_score_table
```

In [74]:
```
final_comparison.sort_values('accuracy_score', ascending=False)
```

Out[74]:

| | True CVD Absent | False CVD Absent | False CVD Present | True CVD Present | accuracy_score |
|---|---|---|---|---|---|
| **Logistic_Regression** | 5505 | 1550 | 2142 | 4540 | 0.731237 |
| **Support_Vector_Machines** | 5744 | 1311 | 2387 | 4295 | 0.730800 |
| **Random_Forest** | 5502 | 1553 | 2186 | 4496 | 0.727815 |
| **KNN** | 5346 | 1709 | 2069 | 4613 | 0.724976 |
| **Naive_Bayes** | 5488 | 1567 | 2968 | 3714 | 0.669870 |
| **Decision_Tree** | 4455 | 2600 | 2496 | 4186 | 0.629031 |

In [ ]: