IoT Heart: Machine Learning

Davide Cantoro

July, 2023

1 Abstract

In recent years, the importance of artificial intelligence, combined with iot devices in the healthcare sector, has been growing. In fact, the use of a digital twin offers an innovative approach in this area and, when used as a monitoring system, can be helpful in determining the medical condition of patients and facilitating quicker decision-making.

The purpose of this paper is to establish the basis for a Machine Learning model that is capable of recognizing people with heart issues and can then be utilized in a Digital Twin in the health care sector.

---

2 Introduction

The dataset used to train the Machine Learning algorithms was taken from Keggle and is the most widely used dataset in the context of Heart Disease. Duplicate observations were first removed because there were not one in the original database from which the Keggle dataset was created.

Data Preprocessing and Data Visualization steps were performed, in order to adjust the dataset and to acquire a knowledge of the distributions and correlations of the features.

Six classification algorithms were selected: Logistic Regression, Decision Tree Classifier, Random Forest Classifier, Support Vector Machines, K-Neighbors Classifier e Naive Bayes.

The dataset was split into a train set and test set (partitioning: 80-20), the train set was used to develop the models while the test set was used to evaluate the model's efficiency at its final stage.

The k-fold cross validation, a resampling technique, was used to train the Machine Learning models. After the models were trained, Tune Hyperparameters was used to identify the best model parameters in order to enhance their performance.

The training of Machine Learning models was performed on two different datasets. - Full Dataset: Dataset containing 10 out of 13 features - Reduced Dataset: is a reduced dataset containing the 4 features used in the digital twin.

The features that will constitute the Full Dataset were chosen using the Extra Trees Classifier, a classification algorithm that is frequently employed in feature selection.

Finally, various metrics were calculated and compared in order to evaluate the performance of the models (differentiating between the two datasets). A comparison with the models produced in the absence of any preliminary data cleaning for duplicates is also included.

---

3 Python libraries used

```
numpy
pandas Dataset manipulation

Plotting
matplotlib
seaborn

Features Selection
sklearn.ensemble -> ExtraTreesClassifier

Split Sataset
sklearn.preprocessing -> MinMaxScaler
sklearn.model_selection -> train_test_split

K-Fold Cross Validation
sklearn -> model_selection

Machine Learning Algorithms
sklearn.linear_model   -> LogisticRegression
sklearn.tree  -> DecisionTreeClassifier
sklearn.ensemble  -> RandomForestClassifier
sklearn.neighbors  -> KNeighborsClassifier
sklearn.naive_bayes   -> GaussianNB
sklearn.svm   -> SVC

 Tune Hyperparameters
sklearn.model_selection -> GridSearchCV

# Machine Learning Report & Metrics
sklearn.metrics -> accuracy_score , precision_score , classification_report

# Confusion Matrix
sklearn.metrics -> confusion_matrix
```

4 Dataset

The dataset considered was taken from Kaggle, it is one of the most widely used datesets in the field of heart disease. The presence of the target variable, which enables the use of Supervised Learning, was another factor in the selection of this dataset.

This data set dates from 1988 and consists of four databases: Cleveland, Hungary, Switzerland, and Long Beach V. It contains 76 attributes, including the predicted attribute, but all published experiments refer to using a subset of 14 of them. The "target" field refers to the presence of heart disease in the patient. It is integer valued 0 = no disease and 1 = disease.

Features	Description	Possible values
age		
sex		1 = Male, 0 = Female
cp	chest pain type	0 = Typical Angina 1 = Atypical Angina, 2 = Non-anginal Pain, 3 = Asymptomatic
trestbps	resting blood pressure, mmHg	
chol	serum cholestoral in mg/dl	
fbs	fasting blood sugar > 120 mg/dl	0 = False, 1 = True
restecg	resting electrocardiographic results	0 = Normal, 1 = ST-T Wave Abnormality, 2 = Showing probable or definite left ventricular hypertrophy
thalach	maximum heart rate achieve	
exang	exercise induced angina	1 = Yes, 0 = No
oldpeak	ST depression induced by exercise relative to rest	
slope	The slope of the peak exercise ST segment	0 = Upsloping, 2 = Flat, 3 = Downsloping
ca	number of major vessels (0-3) colored by flourosopy	
thal	A blood disorder known as thalassaemia	3 = Normal, 6 = Fixed Defect, 7 = Reversible Defect
target	The patient has a heart disease	0 = No, 1 = Yes

trestbps: The optimal blood pressure level is a reading under 120/80 mmHg. A reading that is higher would be considered elevated or high.

chol: The desirable cholesterol level for adults is less than 200 mg/dl.

5 Data Preprocessing

5.1 Rename features

Old Features Name	New Features Name
age	age
sex	sex
cp	chest_pain
trestbps	rest_blood_pressure
chol	cholesterol
fbs	fast_blood_sugar
restecg	rest_ecg
thalach	max_heart_rate
exang	exercise_induced_angina
oldpeak	st_depression
slope	st_slope
ca	num_major_vessels
thal	thalassaemia
target	target

5.2 Remove duplicate observation

As can be seen, the dataset contains a large number of duplicate observations.

These duplicate observations are not in the original "Heart Disease (1988)" database, so the first thing to do is to go and delete the duplicate observations. Duplicate observations are a big problem in the field of Machine Learning as they introduce bias and have an impact on the final model.

In another document (IoT Heart - M.L. with duplicated), the same Machine Learning algorithms done here have been developed, so that the impact they have can be compare.

```
print("There are {} duplicated observations to be remove.".format(len(data[data.
↪duplicated()])))
data.drop_duplicates(keep = 'first', inplace = True)
print("The new dataset will consist of {} observations.".format(len(data.index)))
```

```
There are 723 duplicated observations to be remove.
The new dataset will consist of 302 observations.
```

5.3 Features type

```
data.dtypes
```

```
age                          int64
sex                          int64
chest_pain                   int64
rest_blood_pressure          int64
cholesterol                  int64
fast_blood_sugar             int64
rest_ecg                     int64
max_heart_rate               int64
exercise_induced_angina      int64
st_depression              float64
st_slope                     int64
num_major_vessels            int64
thalassaemia                 int64
target                       int64
dtype: object
```

5.4 Looking for null values

```
#check for null value in dataset
data.isnull().sum()
```

```
age                        0
sex                        0
chest_pain                 0
rest_blood_pressure        0
cholesterol                0
fast_blood_sugar           0
rest_ecg                   0
max_heart_rate             0
exercise_induced_angina    0
st_depression              0
st_slope                   0
num_major_vessels          0
thalassaemia               0
target                     0
dtype: int64
```

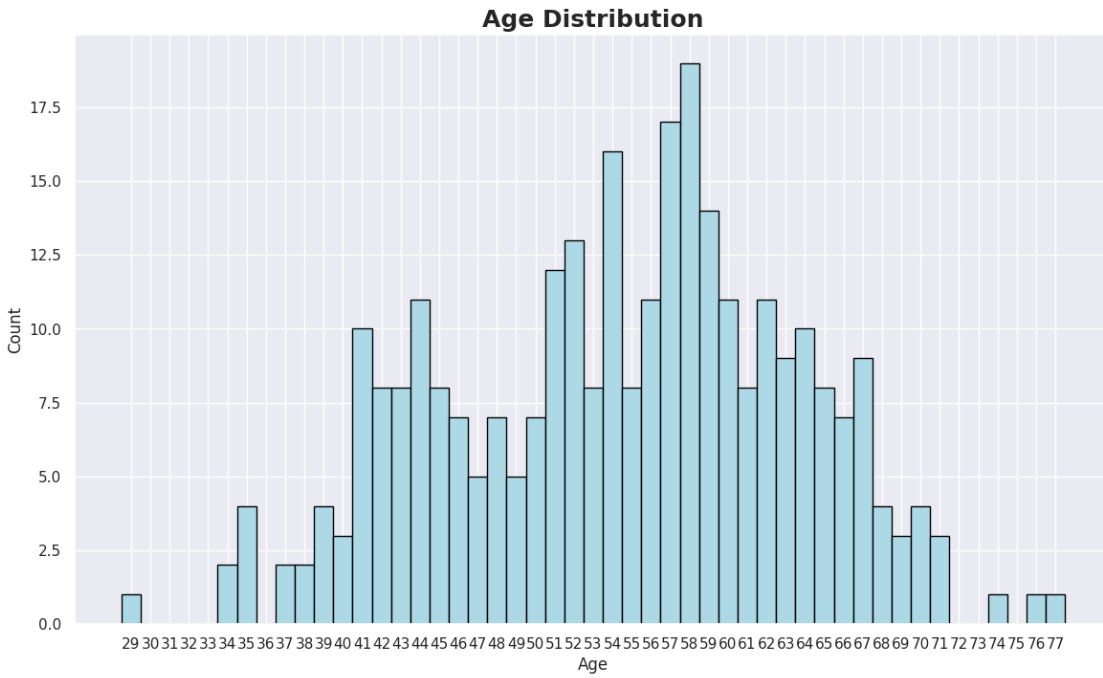
6 Data Visualization

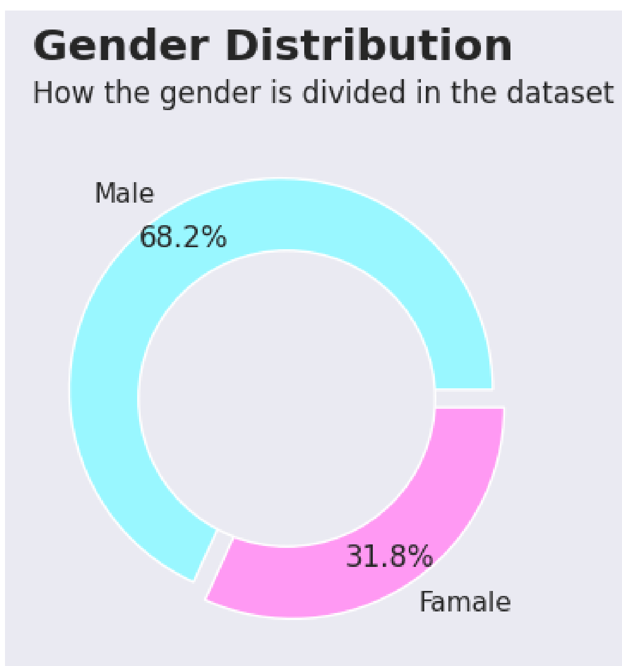
6.1 Statistical info

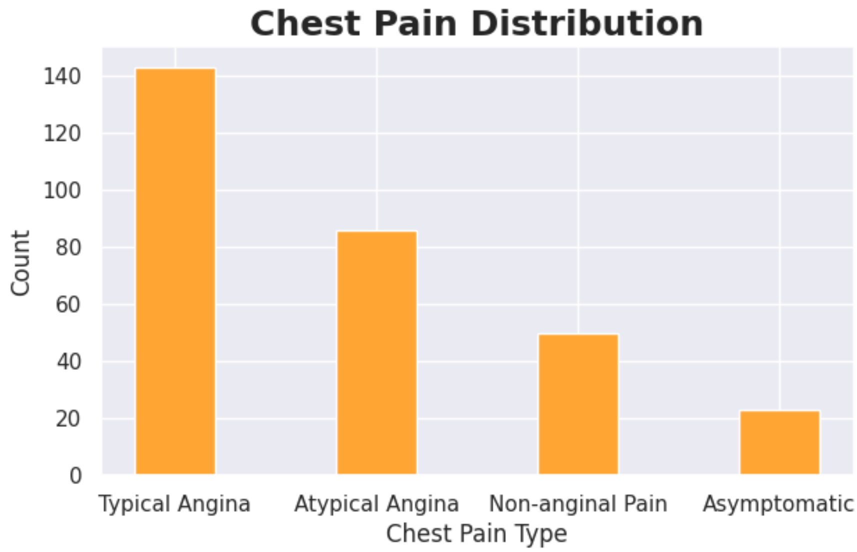
```python
#Statistical info
data.describe()
```

```
              age         sex  chest_pain  rest_blood_pressure  cholesterol  \
count  302.00000  302.000000  302.000000           302.000000   302.000000
mean    54.42053    0.682119    0.963576           131.602649   246.500000
std      9.04797    0.466426    1.032044            17.563394    51.753489
min     29.00000    0.000000    0.000000            94.000000   126.000000
25%     48.00000    0.000000    0.000000           120.000000   211.000000
50%     55.50000    1.000000    1.000000           130.000000   240.500000
75%     61.00000    1.000000    2.000000           140.000000   274.750000
max     77.00000    1.000000    3.000000           200.000000   564.000000

       fast_blood_sugar    rest_ecg  max_heart_rate  exercise_induced_angina  \
count        302.000000  302.000000      302.000000               302.000000
mean           0.149007    0.526490      149.569536                 0.327815
std            0.356686    0.526027       22.903527                 0.470196
min            0.000000    0.000000       71.000000                 0.000000
25%            0.000000    0.000000      133.250000                 0.000000
50%            0.000000    1.000000      152.500000                 0.000000
75%            0.000000    1.000000      166.000000                 1.000000
max            1.000000    2.000000      202.000000                 1.000000

       st_depression    st_slope  num_major_vessels  thalassaemia      target
count     302.000000  302.000000         302.000000    302.000000  302.000000
mean        1.043046    1.397351           0.718543      2.314570    0.543046
std         1.161452    0.616274           1.006748      0.613026    0.498970
min         0.000000    0.000000           0.000000      0.000000    0.000000
25%         0.000000    1.000000           0.000000      2.000000    0.000000
50%         0.800000    1.000000           0.000000      2.000000    1.000000
75%         1.600000    2.000000           1.000000      3.000000    1.000000
max         6.200000    2.000000           4.000000      3.000000    1.000000
```

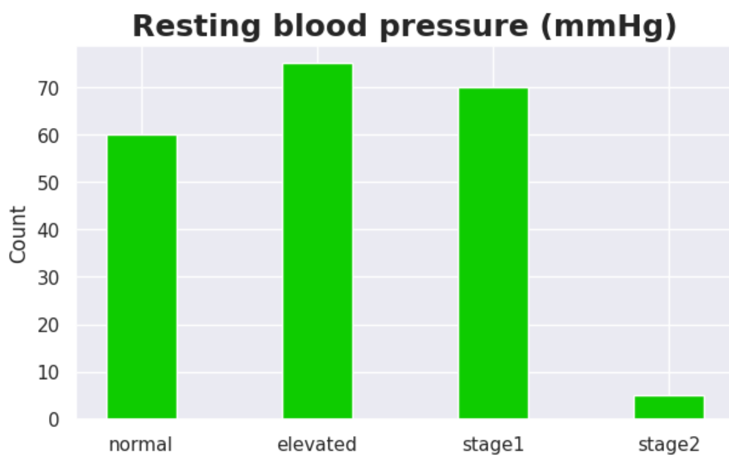
6.2 Distribution





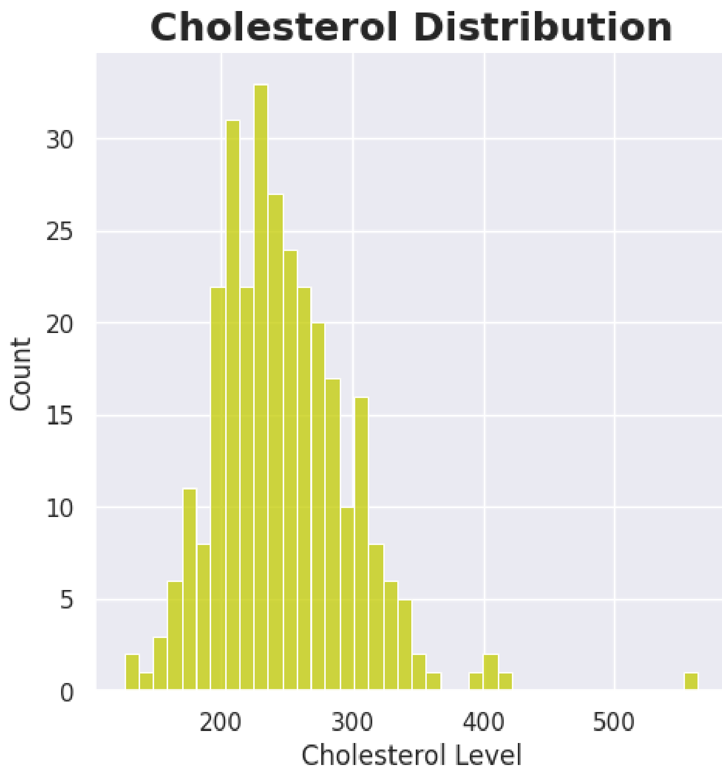


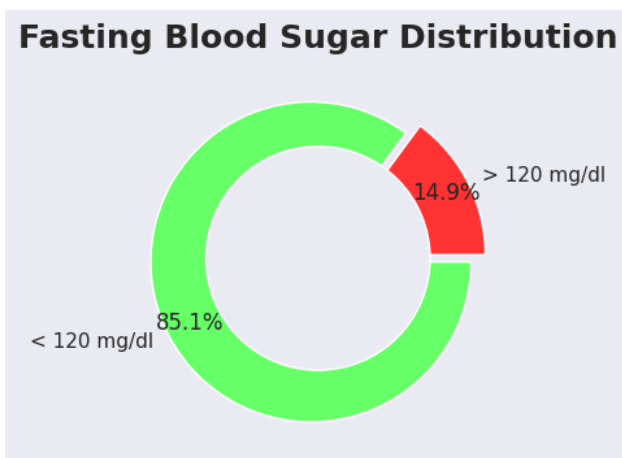

```
    Chest Pain Type  Count
0     Typical Angina    143
1    Atypical Angina     86
2   Non-anginal Pain     50
3       Asymptomatic     23
```



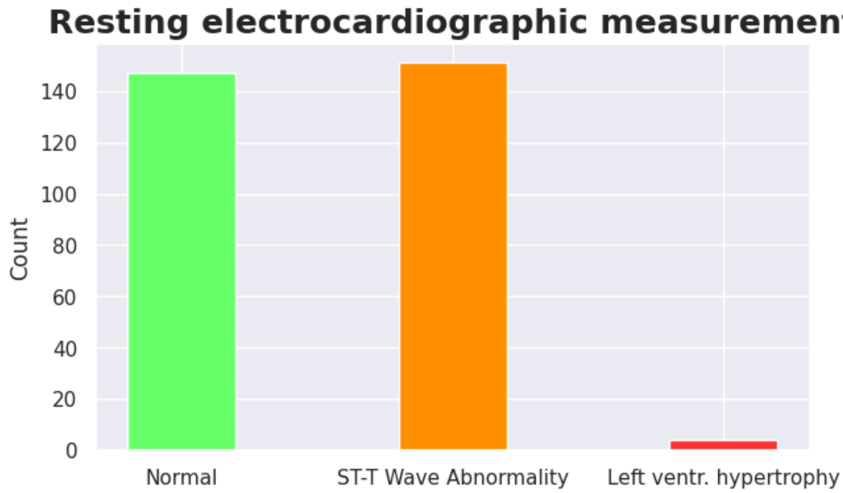
Resting blood pressure (mmHg)

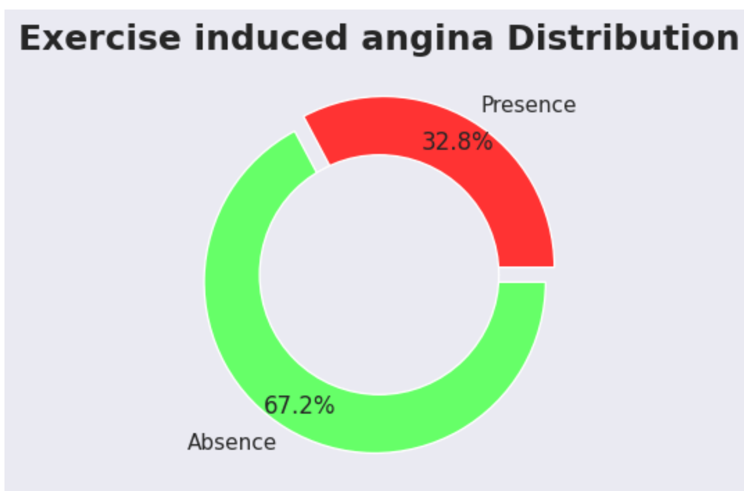
Blood Pressure Category	Systolic Blood Pressure		Diastolic Blood Pressure
Normal	<120 mmHg	and	<80 mmHg
Elevated	120-129 mmHg	and	<80 mmHg
Hypertension			
Stage 1	130-139 mmHg	or	80-89 mmHg
Stage 2	>=140 mmHg	or	>=90 mmHg



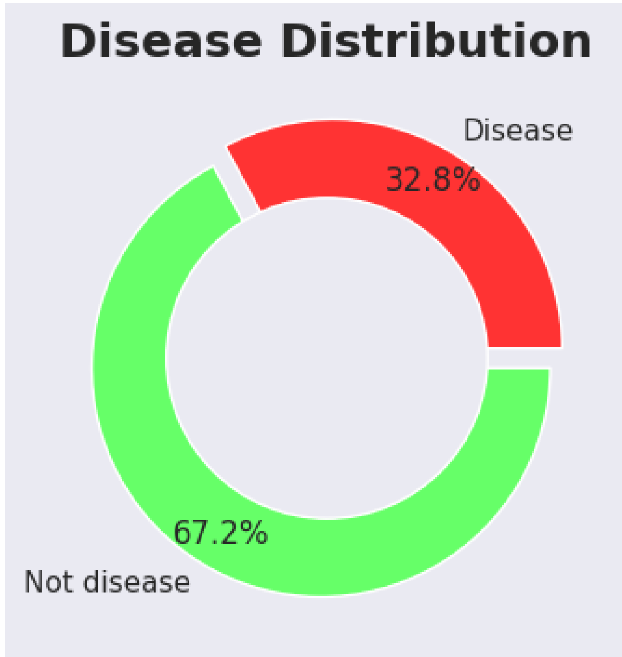


	F.B.S.	Count
0	> 120 mg/dl	45
1	< 120 mg/dl	257





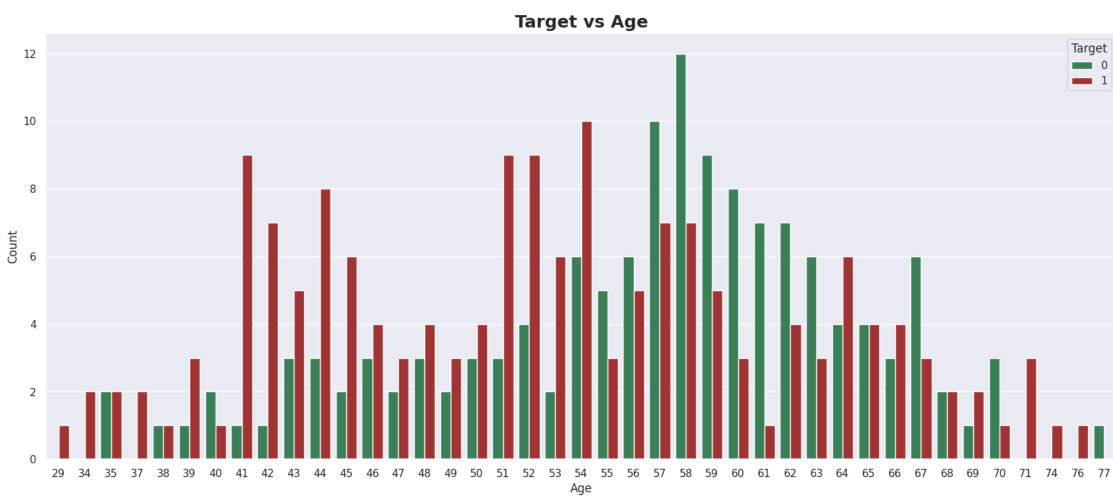
	Exang	Count
0	Presence	99
1	Absence	203

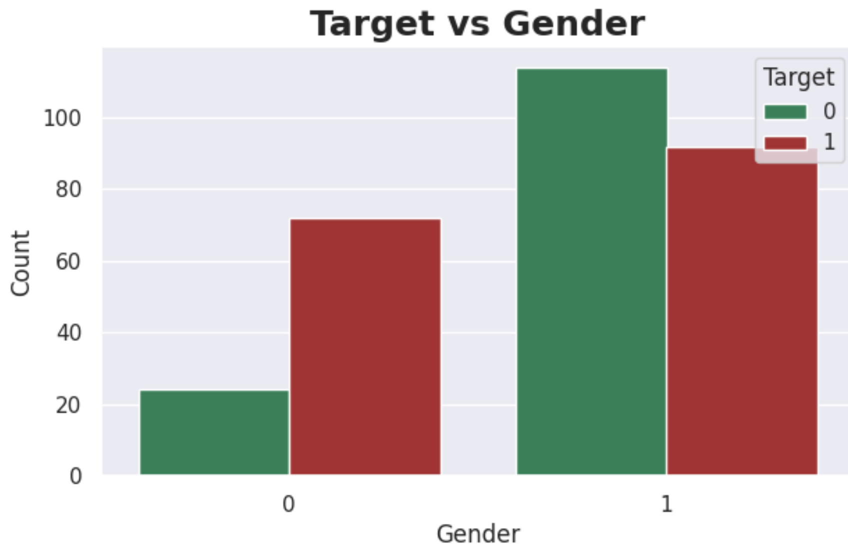


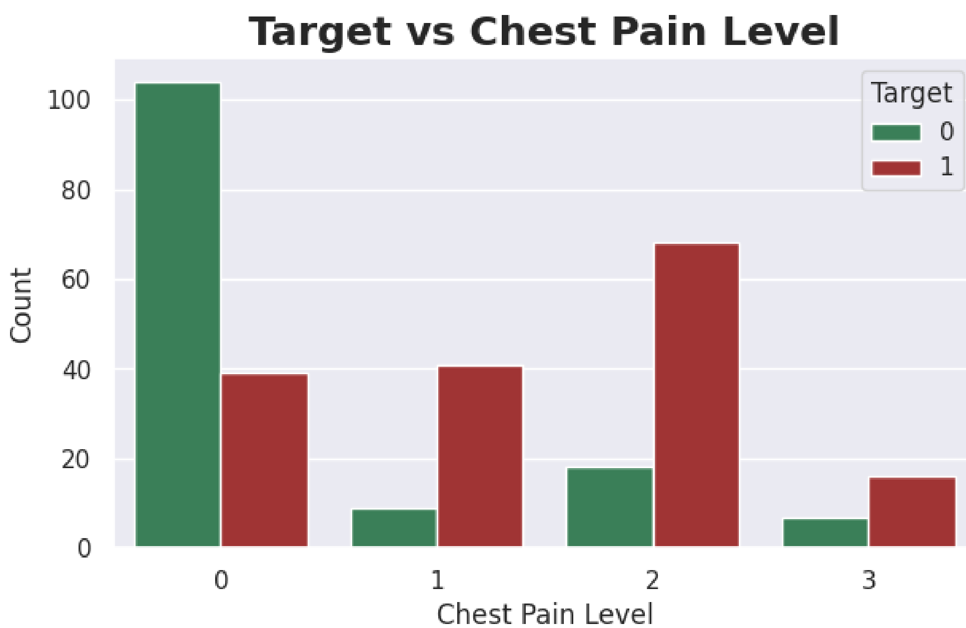
		Count
0	Disease	99
1	Not disease	203

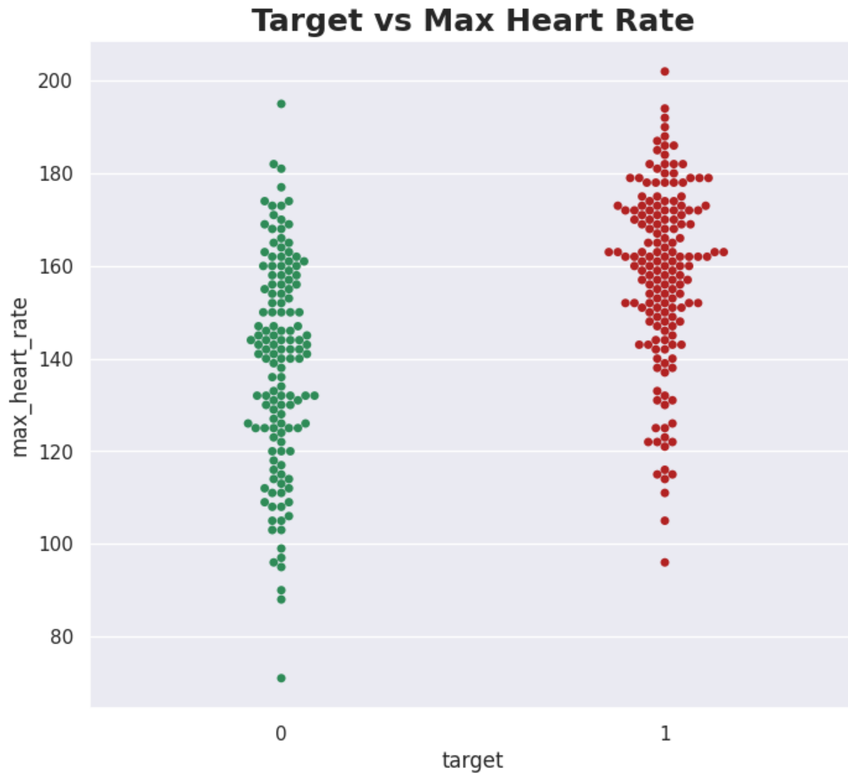
6.3 Features Selection and Bivariate Analysis

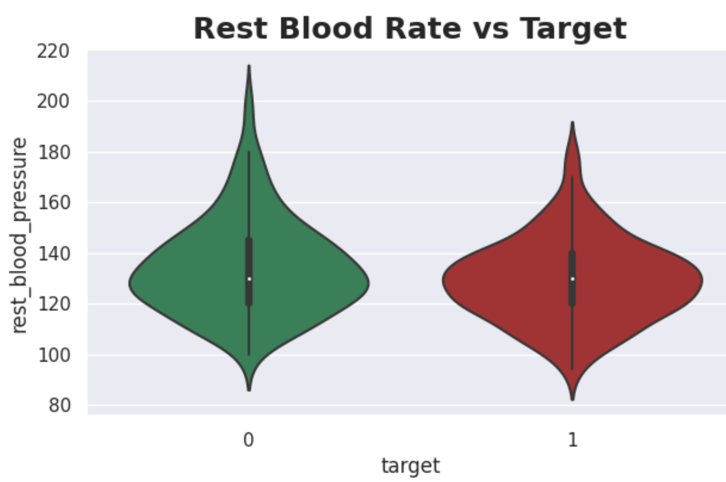
6.3.1 Correlation with target




Target vs Gender
Target
0
1
100
80
60
40
20
0
Count
0
1
Gender


Target vs Chest Pain Level
Target
0
1
100
80
60
40
20
0
Count
0
1
2
3
Chest Pain Level

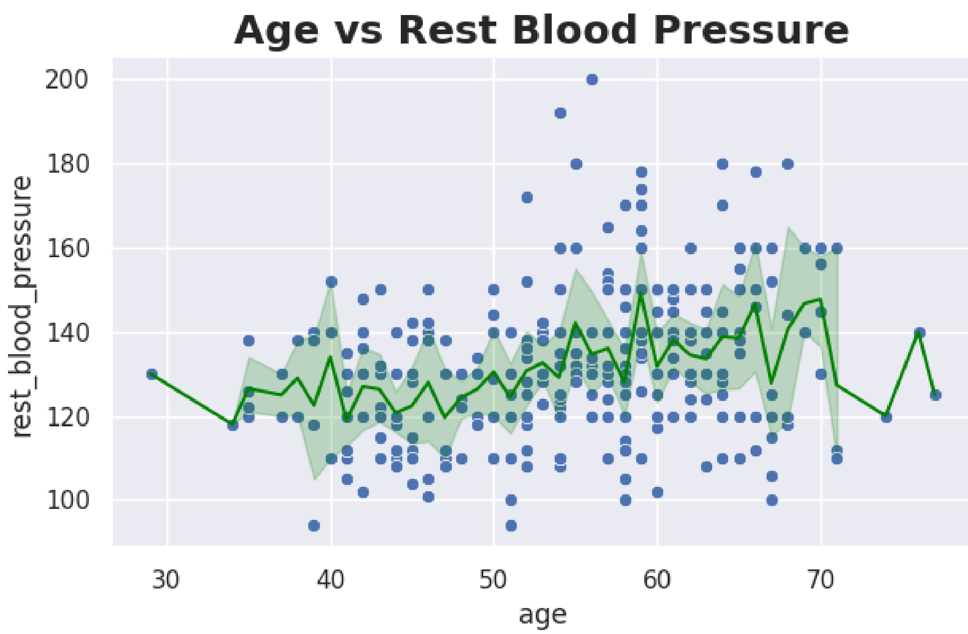


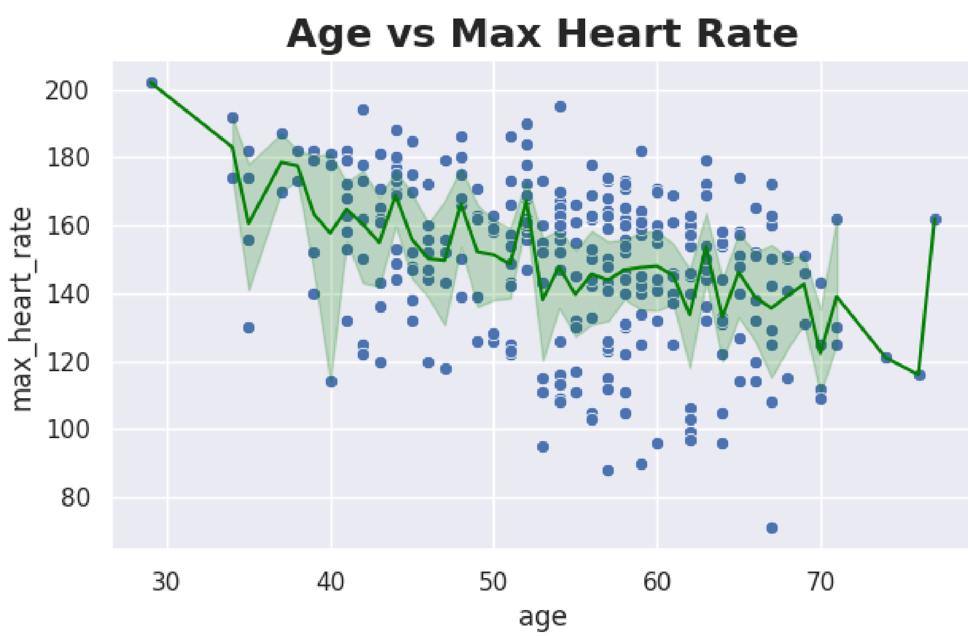


According to WebMD, a high blood pressure is an indication of heart disease, the problem is that looking at the violin plot we cannot state this.

source: https://www.webmd.com/hypertension-high-blood-pressure/hypertensive-heart-disease

6.3.2 Correletion between features

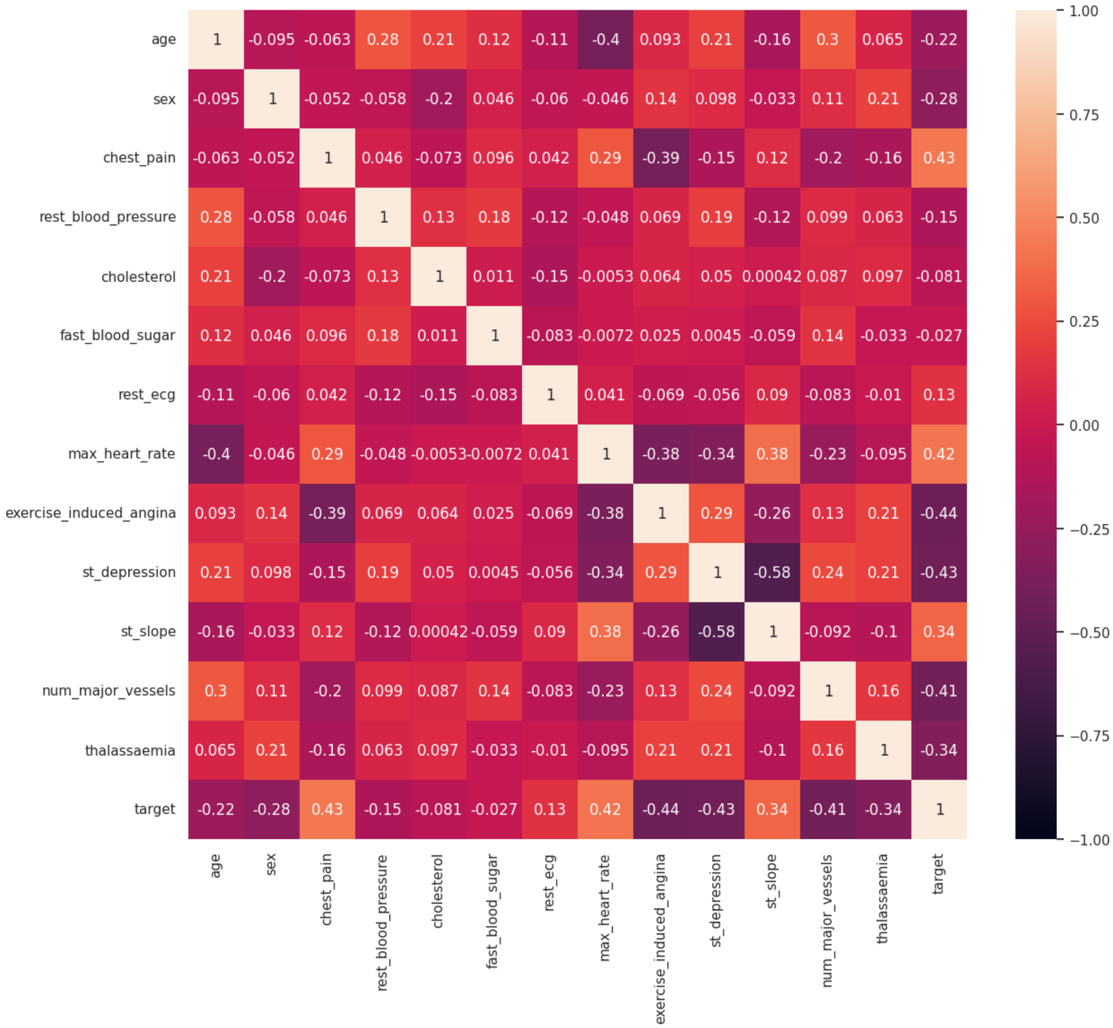




7 Model building

7.1 Feature selection

7.1.1 Correlation index (Heatmap)



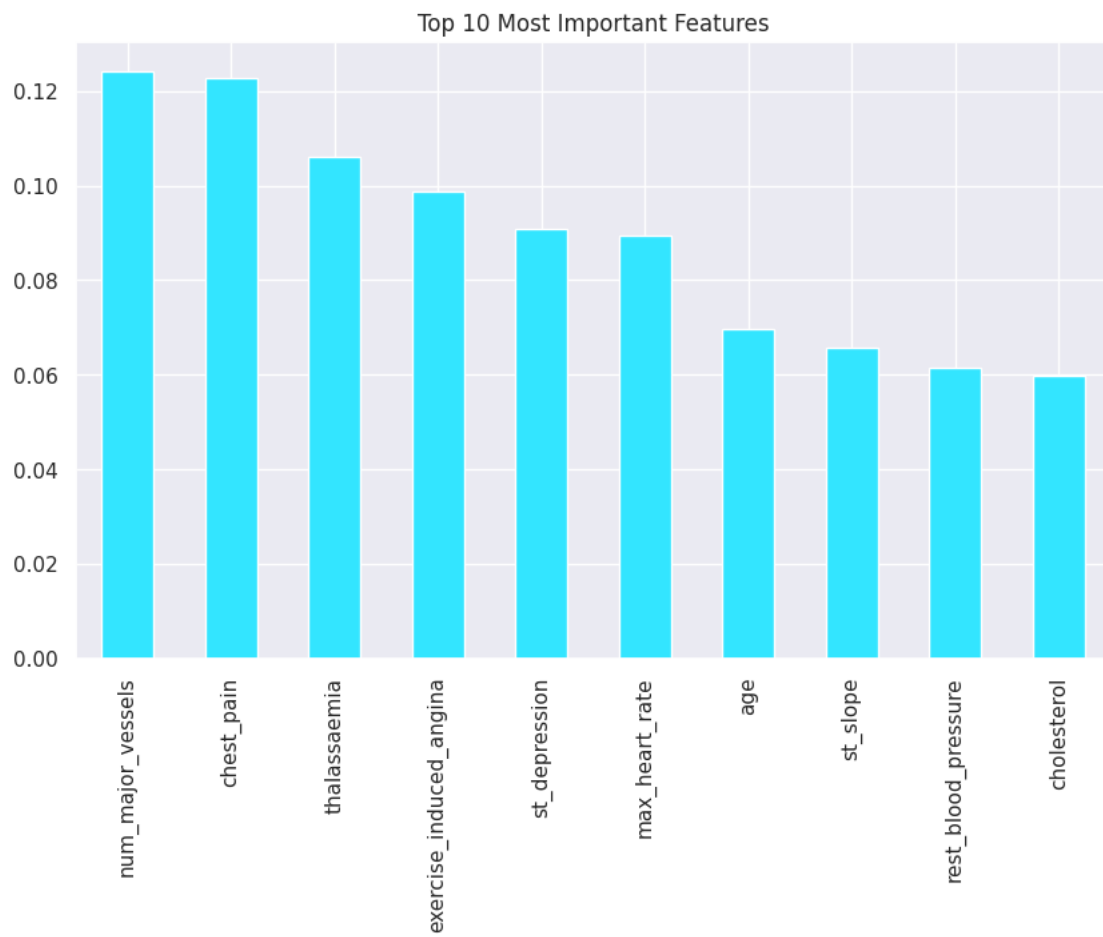
7.1.2 Extra Trees Classifier

The Extra Trees Classifier is similar to the Random Forest Classifier (the difference is in the way the tree is constructed). It's an ensemble learning technique useful for feature selection (it uses the Gini Index as a criterion for measuring correlation).

[Code]:
```
X = data.iloc[:,0:13]
Y = data.iloc[:,-1] # target column

# Building the model
extratrees = ExtraTreesClassifier(n_estimators=250 , random_state=0) #␣
↪n_estimators = # of tree

# Training the model
extratrees.fit(X,Y)

#plot setting
sn.set(style="darkgrid")
plt.figure(figsize=(10,6))
plt.title("Top 10 Most Important Features")

#to get the first 10 important for plotting
feat_importances = pd.Series(extratrees.feature_importances_, index=X.columns)
feat_importances.nlargest(10).plot(kind='bar', color='#33e5ff')

# Computing the importance of each feature
importance = extratrees.feature_importances_

feature_labels=np.array(X.columns)

#sort by importance
feature_indexes_by_importance = importance.argsort()

#print (all feature) in % form, descend
for index in feature_indexes_by_importance:
    print(" {} - {:2f}%".format( feature_labels[index], (importance[index] * 100.
↪0)))

plt.show()
```

Output:

```
fast_blood_sugar - 1.982229%
rest_ecg - 3.539538%
sex - 5.638410%
cholesterol - 5.974379%
rest_blood_pressure - 6.145757%
st_slope - 6.581179%
age - 6.967207%
max_heart_rate - 8.939465%
st_depression - 9.071878%
exercise_induced_angina - 9.867437%
thalassaemia - 10.616395%
chest_pain - 12.263716%
num_major_vessels - 12.412410%
```



7.2 Dataset preparation

7.3 Split dataset

[Code]:
```
y = data['target']
x = data.drop('target' ,axis=1)

scaler = MinMaxScaler()

X = pd.DataFrame(scaler.fit_transform(x) , columns = x.columns)

X_train_origin , X_test_origin , Y_train , Y_test = train_test_split(X,y␣
↪,train_size=0.8) #80% train - 20% test
```

7.4 Create new dataset with selected features

[Code]:
```
X_train = X_train_origin[['age', 'chest_pain', 'rest_blood_pressure',␣
↪'cholesterol', 'max_heart_rate',
       'exercise_induced_angina', 'st_depression', 'st_slope',␣
↪'num_major_vessels', 'thalassaemia']]
X_test = X_test_origin[['age', 'chest_pain', 'rest_blood_pressure',␣
↪'cholesterol', 'max_heart_rate',
       'exercise_induced_angina', 'st_depression', 'st_slope',␣
↪'num_major_vessels', 'thalassaemia']]
```

8 Model

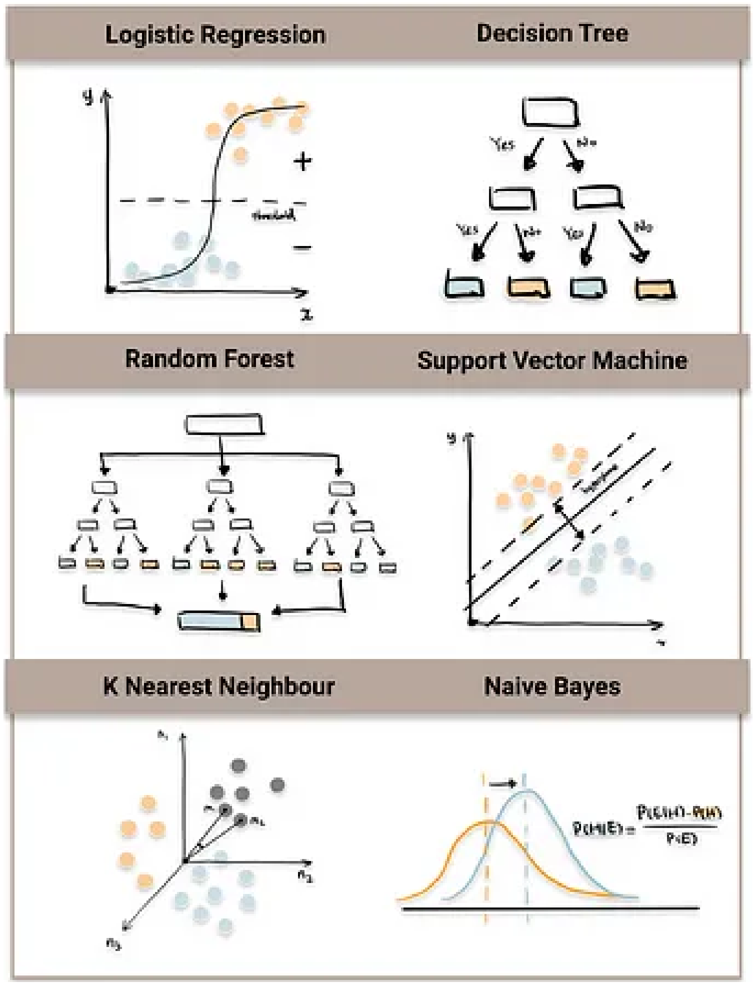


Image by Destin Gong on towardsdatascience.com

Selected Algorithms:

1. Logistic Regression (LR)
2. Decision Tree Classifier (DTC)
3. Random Forest Classifier (RFC)
4. Support Vector Machines (SVM)
5. K-Neighbors Classifier (KNN)
6. Naive Bayes (NB)

8.1 K-fold Cross Validation

Cross-validation is a resampling technique that uses various data subsets to test and train a model on various iterations.

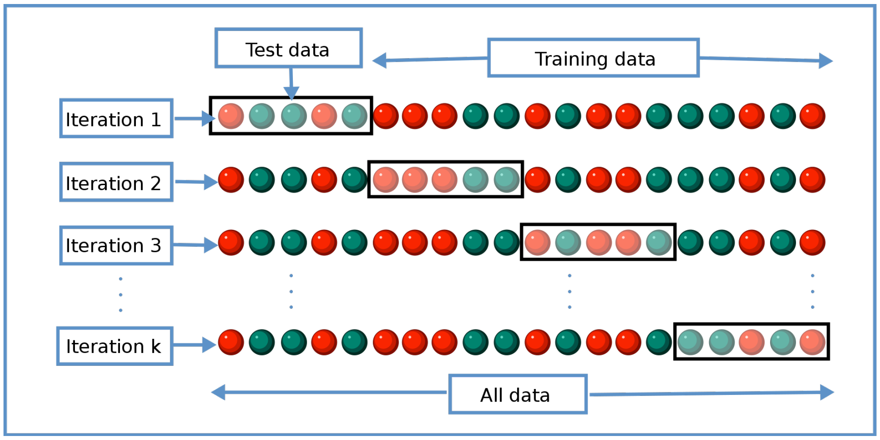


Image by Gufosowa on en.wikipedia.org

[Code]:
```
# Classification models
models = []
models.append(('LR', LogisticRegression(solver='lbfgs', max_iter=5000)))
models.append(('DTC', DecisionTreeClassifier()))
models.append(('RFC', RandomForestClassifier(n_estimators=100)))
models.append(('SVM', SVC(gamma='scale')))
models.append(('KNN', KNeighborsClassifier()))
models.append(('NB', GaussianNB()))
# evaluate accuracy for each model
results = []
names = []
for name, model in models:
    kfold = model_selection.KFold(n_splits=10)
    cv_results = model_selection.cross_val_score(model, X_train, Y_train,␣
 ↪cv=kfold, scoring="accuracy")
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
LR: 0.816833 (0.084032)
DTC: 0.734667 (0.058137)
RFC: 0.779667 (0.079784)
SVM: 0.787833 (0.068973)
KNN: 0.804500 (0.065269)
NB: 0.804500 (0.079645)
```

8.2 Tune Hyperparameters

The M.L. algorithms used so far refer to default parameters (using a grid approach), to improve performance and search for the optimal model one can use Tune Hyperparameters.

The parameter tune must be done on a single model at a time and consists of running the model several times going (varying the parameters) until the best one is found..

8.2.1 Logistic Regression

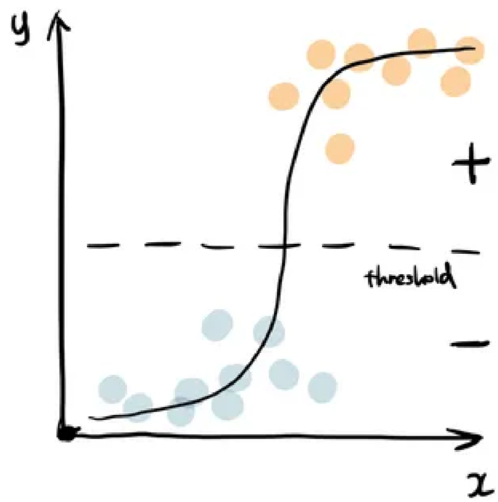


Image by Destin Gong on towardsdatascience.com

[Code]:
```
# Create the model
logreg = LogisticRegression(random_state=0, max_iter=5000, solver='liblinear')
#Parameters for grid search
solvers = ['newton-cg', 'lbfgs', 'liblinear']
penalty = ['l2']
c_values = [100, 10, 1.0, 0.1, 0.01]
param_grid = dict(solver=solvers,penalty=penalty,C=c_values)
grid = GridSearchCV(estimator=logreg, param_grid=param_grid, cv=5,␣
↪scoring="accuracy", verbose=1, n_jobs=4) # cv = 5 to use the default 5-fold␣
↪cross validation

# Run grid search on the training data
grid.fit(X_train,Y_train)

# print score
print("Accuracy: ",grid.best_score_)
print("Model: ",grid.best_estimator_)
print("Best parameters: ",grid.best_params_)

LogisticRegression_best_model = grid.best_estimator_
```

```
Fitting 5 folds for each of 15 candidates, totalling 75 fits
Accuracy:  0.8215986394557824
Model:  LogisticRegression(C=100, max_iter=5000, random_state=0,
solver='liblinear')
Best parameters:  {'C': 100, 'penalty': 'l2', 'solver': 'liblinear'}
```

8.2.2 Decision Tree

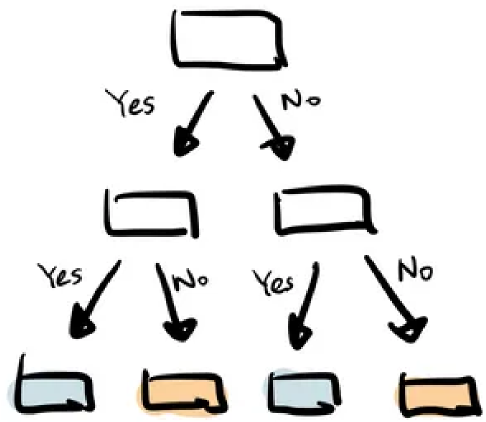


Image by Destin Gong on towardsdatascience.com

[Code]:
```
# Create the model
decision_tree = DecisionTreeClassifier(random_state=0)

#Parameters for grid search
max_depth = [3, 6, 10, 20, 30, 50]
max_features = [1.0, 0.5, 0.1]
min_samples_leaf = [3, 6, 9, 12, 15, 30]
min_samples_split = [2, 4, 8, 15, 25]
param_grid = dict(max_depth=max_depth,max_features=max_features,
   min_samples_leaf=min_samples_leaf,min_samples_split=min_samples_split)

grid = GridSearchCV(estimator=decision_tree, param_grid=param_grid, cv=5,␣
 ↪scoring="accuracy", verbose=1, n_jobs=4) # cv = 5 to use the default 5-fold␣
 ↪cross validation

# Run grid search on the training data
grid.fit(X_train,Y_train)

# print score
print("Accuracy: ",grid.best_score_)
print("Model: ",grid.best_estimator_)
print("Best parameters: ",grid.best_params_)
DecisionTreeClassifier_best_model = grid.best_estimator_
```

```
Fitting 5 folds for each of 540 candidates, totalling 2700 fits
Accuracy:  0.7843537414965986
Model:  DecisionTreeClassifier(max_depth=10, max_features=0.5,
min_samples_leaf=3,
                       min_samples_split=8, random_state=0)
Best parameters:  {'max_depth': 10, 'max_features': 0.5, 'min_samples_leaf': 3,
'min_samples_split': 8}
```

8.2.3 Random Forest

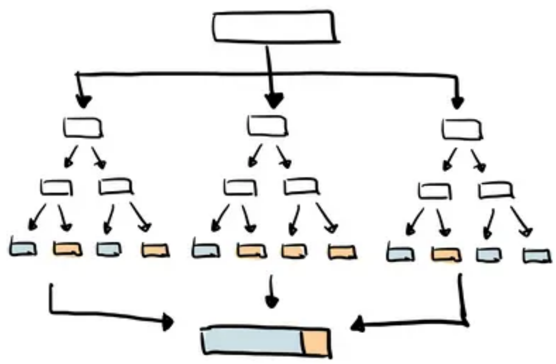


Image by Destin Gong on towardsdatascience.com

[Code]:
```
# Create the model
forest = RandomForestClassifier(random_state=0)

#Parameters for grid search
n_estimators = [200, 300, 500, 750]
max_features = ['sqrt', 'log2']
param_grid = dict(n_estimators=n_estimators,max_features=max_features)


grid = GridSearchCV(estimator=forest, param_grid=param_grid, cv=5,␣
↪scoring="accuracy", verbose=1, n_jobs=4) # cv = 5 to use the default 5-fold␣
↪cross validation

# Run grid search on the training data
grid.fit(X_train,Y_train)

# print score
print("Accuracy: ",grid.best_score_)
print("Model: ",grid.best_estimator_)
print("Best parameters: ",grid.best_params_)


RandomForestClassifier_best_model = grid.best_estimator_
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Accuracy:  0.8008503401360544
Model:  RandomForestClassifier(n_estimators=300, random_state=0)
Best parameters:  {'max_features': 'sqrt', 'n_estimators': 300}
```

8.2.4 Support Vector Machine (SVM)

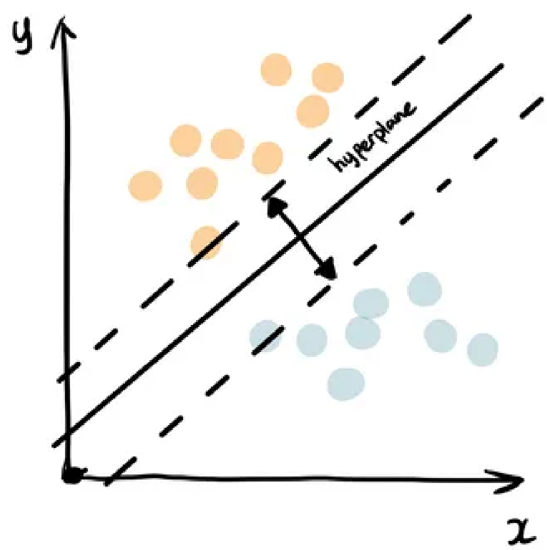


Image by Destin Gong on towardsdatascience.com

[Code]:
```
# Create the model
svm = SVC()
#Parameters for grid search
kernel = ['poly', 'rbf', 'sigmoid']
C = [50, 10, 1.0, 0.1, 0.01]
gamma = ['scale']
param_grid = dict(kernel=kernel,C=C,gamma=gamma)
grid = GridSearchCV(estimator=svm, param_grid=param_grid, cv=5,␣
 ↪scoring="accuracy", verbose=1, n_jobs=4) # cv = 5 to use the default 5-fold␣
 ↪cross validation

# Run grid search on the training data
grid.fit(X_train,Y_train)
# print score
print("Accuracy: ",grid.best_score_)
print("Model: ",grid.best_estimator_)
print("Best parameters: ",grid.best_params_)
SVC_best_model = grid.best_estimator_
```

```
Fitting 5 folds for each of 15 candidates, totalling 75 fits
Accuracy:  0.813265306122449
Model:  SVC(C=0.1, kernel='poly')
Best parameters:  {'C': 0.1, 'gamma': 'scale', 'kernel': 'poly'}
```

8.2.5 K-Nearest Neighbour (KNN)

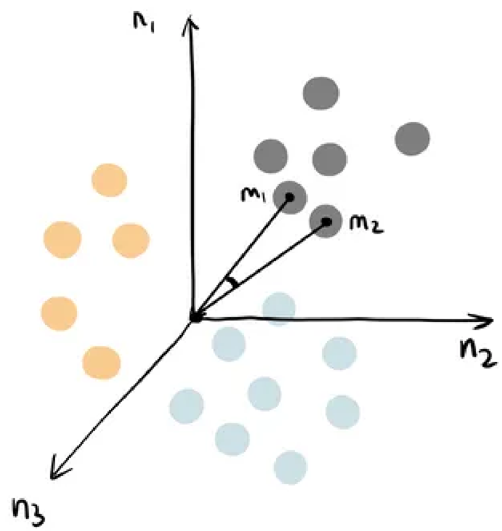


Image by Destin Gong on towardsdatascience.com

[Code]:
```
# Create the model
knn = KNeighborsClassifier()

#Parameters for grid search
n_neighbors = [1, 50]
weights = ['uniform', 'distance']
metric = ['euclidean', 'manhattan', 'minkowski', 'chebyshev']
param_grid = dict(n_neighbors=n_neighbors,weights=weights,metric=metric)
grid = GridSearchCV(estimator=knn, param_grid=param_grid, cv=5,␣
 ↪scoring="accuracy", verbose=1, n_jobs=4) # cv = 5 to use the default 5-fold␣
 ↪cross validation

# Run grid search on the training data
grid.fit(X_train,Y_train)

# print score
print("Accuracy: ",grid.best_score_)
print("Model: ",grid.best_estimator_)
print("Best parameters: ",grid.best_params_)

KNeighborsClassifier_best_model = grid.best_estimator_
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
Accuracy:  0.8046768707482993
Model:  KNeighborsClassifier(metric='manhattan', n_neighbors=50,
weights='distance')
Best parameters:  {'metric': 'manhattan', 'n_neighbors': 50, 'weights':
'distance'}
```

8.2.6

Naive Bayes

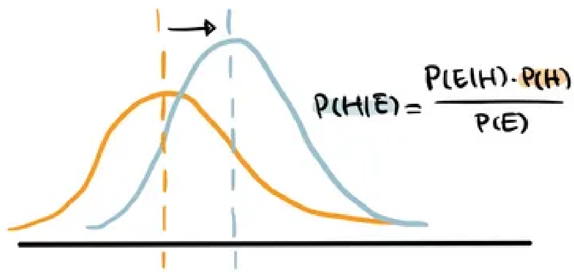


Image by Destin Gong on towardsdatascience.com

[Code]:
```
# Create the model
gaussNB = GaussianNB()

#Parameters for grid search
var_smoothing= np.logspace(0,-9, num=100)
param_grid = dict(var_smoothing=var_smoothing)
grid = GridSearchCV(estimator=gaussNB, param_grid=param_grid, cv=5,␣
↪scoring="accuracy", verbose=1, n_jobs=4) # cv = 5 to use the default 5-fold␣
↪cross validation

# Run grid search on the training data
grid.fit(X_train,Y_train)

# print score
print("Accuracy: ",grid.best_score_)
print("Model: ",grid.best_estimator_)
print("Best parameters: ",grid.best_params_)


GaussianNB_best_model = grid.best_estimator_
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
Accuracy:  0.8214285714285714
Model:  GaussianNB(var_smoothing=0.012328467394420659)
Best parameters:  {'var_smoothing': 0.012328467394420659}
```

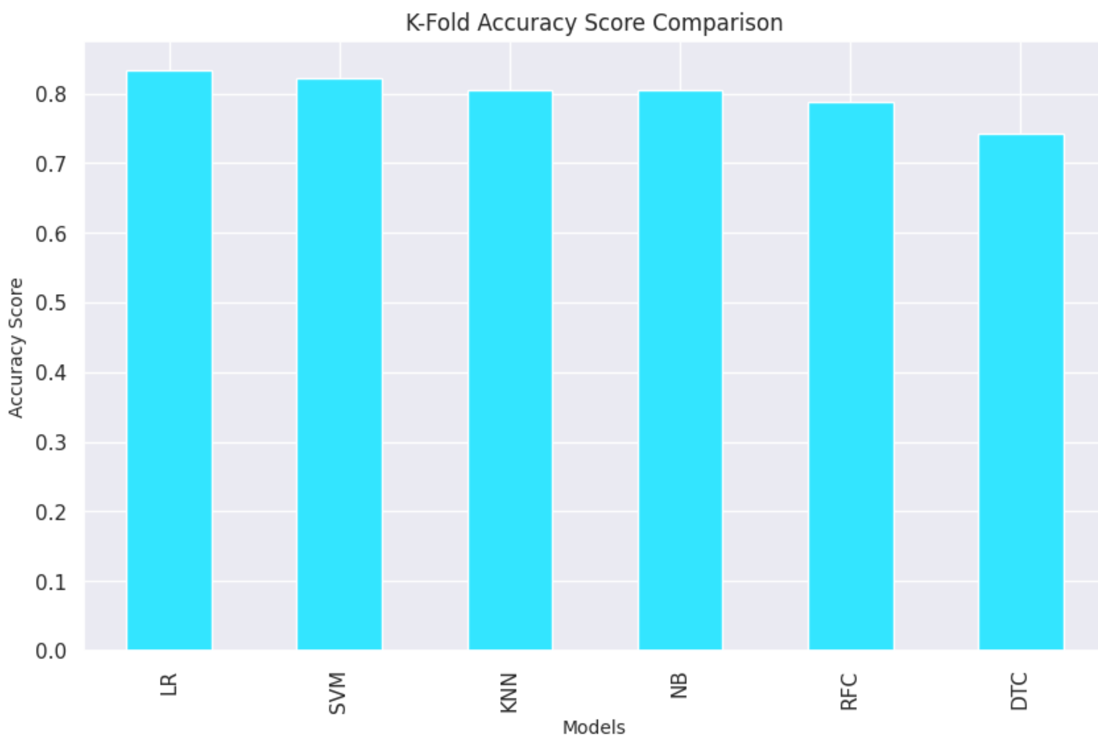
8.3 Repeat k-Fold Cross Validation with Tuned Hyperparameters

[Code]:
```
# Classification models
models = []
models.append(('LR', LogisticRegression_best_model))
models.append(('DTC', DecisionTreeClassifier_best_model))
models.append(('RFC', RandomForestClassifier_best_model))
models.append(('SVM', SVC_best_model))
models.append(('KNN', KNeighborsClassifier_best_model))
models.append(('NB', GaussianNB_best_model))


# evaluate accuracy for each model
tune_results = []
for name, model in models:
    kfold = model_selection.KFold(n_splits=10)
    cv_results = model_selection.cross_val_score(model, X_train, Y_train,␣
 ↪cv=kfold, scoring="accuracy")
    tune_results.append(cv_results)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
LR: 0.833500 (0.083584)
DTC: 0.743167 (0.066798)
RFC: 0.788000 (0.058060)
SVM: 0.821500 (0.046058)
KNN: 0.800333 (0.100393)
NB: 0.804500 (0.070388)
```

Print Improvement:

```
        Default model        Tuned model           Improvement
LR:     0.816833 (0.084032)  0.833500 (0.084032)   0.016666666666666607
DTC:    0.734667 (0.058137)  0.743167 (0.058137)   0.008500000000000063
RFC:    0.779667 (0.079784)  0.788000 (0.079784)   0.008333333333333304
SVM:    0.787833 (0.068973)  0.821500 (0.068973)   0.03366666666666662
KNN:    0.804500 (0.065269)  0.800333 (0.065269)   -0.004166666666666652
NB:     0.804500 (0.079645)  0.804500 (0.079645)   0.0

For the following Model there will be use the default parameters:  ['KNN', 'NB']
```



9 Best Machine Learning Model

9.1 Model

```
Best model is LR with an accuracy score of 83.35000000000001%
```

```
LogisticRegression
LogisticRegression(C=100, max_iter=5000, random_state=0, solver='liblinear')
```

[Code]:
```
# If the selected model uses basic parameters, the model must be fit
best_model_name = feat_importances.nlargest(1).index[0]

if best_model_name in default_parameter: # is a vanilla model
  best_model.fit(X_train,Y_train)
```

9.2 Classification Report

```
              precision    recall  f1-score   support

           0       0.76      0.67      0.71        24
           1       0.80      0.86      0.83        37

    accuracy                           0.79        61
   macro avg       0.78      0.77      0.77        61
weighted avg       0.79      0.79      0.78        61
```

9.3 Accuracy, Prediction, Missclassification Rate

```
Accuracy: 0.7868852459016393

Precision: 0.8

Misclassification Rate: 0.21311475409836067
```

9.4 Confusion Matrix, False Positive, False Negative

```
    Confusion Matrix
    [[16  8]
     [ 5 32]]

There are  8  False Positive with rate of  0.3333333333333333
There are  5  False Negative with rate of  0.13513513513513514
```

10 Reduced Dataset

Because these features were taken into consideration when developing the Digital Twin, it was decided to use this smaller dataset.

Since the Reduced Dataset features are the most easily to acquire given the context of the Use Case, they have been taken into consideration for the first prototype of the Digital Twin.

The new Dataset will be composed as follows:

Features	Description	Possible values
age		
sex		1 = Male, 0 = Female
chest_pain	chest pain type	0 = Typical Angina 1 = Atypical Angina, 2 = Non-anginal Pain, 3 = Asymptomatic
max_heart_rate	maximum heart rate achieve	
target	The patient has a heart disease	0 = No, 1 = Yes

10.1 Dataset Preparation

[Code]:
```
X_train = X_train_origin[['age','sex','chest_pain','max_heart_rate']]
X_test = X_test_origin[['age','sex','chest_pain','max_heart_rate']]
```

10.2 Find Best Model

For the Reduced Dataset, the same operations were performed as in Chapter 8

```
K-Fold Cross Validation for a reduced dataset
```

```
LR: 0.754333 (0.106469)
DTC: 0.680333 (0.089910)
RFC: 0.750500 (0.100979)
SVM: 0.766833 (0.115447)
KNN: 0.779500 (0.112294)
NB: 0.754500 (0.105021)
```

```
Tune Hyperparameters
```

[LR]:

```
Fitting 5 folds for each of 15 candidates, totalling 75 fits
Accuracy:  0.755017006802721
Model:  LogisticRegression(C=0.01, max_iter=5000, random_state=0,
solver='liblinear')
Best parameters:  {'C': 0.01, 'penalty': 'l2', 'solver': 'liblinear'}
```

[DT]:

```
Fitting 5 folds for each of 540 candidates, totalling 2700 fits
Accuracy:  0.7760204081632652
Model:  DecisionTreeClassifier(max_depth=6, max_features=0.1,
min_samples_leaf=6,
                       min_samples_split=25, random_state=0)
Best parameters:  {'max_depth': 6, 'max_features': 0.1, 'min_samples_leaf': 6,
'min_samples_split': 25}
```

[RF]:

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Accuracy:  0.7634353741496598
Model:  RandomForestClassifier(n_estimators=300, random_state=0)
Best parameters:  {'max_features': 'sqrt', 'n_estimators': 300}
```

[SVC]:

```
Fitting 5 folds for each of 15 candidates, totalling 75 fits
Accuracy:  0.7800170068027211
Model:  SVC(C=50)
Best parameters:  {'C': 50, 'gamma': 'scale', 'kernel': 'rbf'}
```

[KNN]:

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
Accuracy:  0.7633503401360543
Model:  KNeighborsClassifier(metric='manhattan', n_neighbors=50,
weights='distance')
Best parameters:  {'metric': 'manhattan', 'n_neighbors': 50, 'weights':
'distance'}
```

[NB]:

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
Accuracy:  0.7631802721088435
Model:  GaussianNB(var_smoothing=0.02848035868435802)
Best parameters:  {'var_smoothing': 0.02848035868435802}
```

```
K-Fold Cross Validation with Tuned Hyperparameters for Reduced Dataset
```

```
        Default model           Tuned model             Improvement
LR:     0.754333 (0.106469)     0.833500 (0.106469)     0.07916666666666665
DTC:    0.680333 (0.089910)     0.743167 (0.089910)     0.06283333333333333
RFC:    0.750500 (0.100979)     0.788000 (0.100979)     0.03750000000000009
SVM:    0.766833 (0.115447)     0.821500 (0.115447)     0.05466666666666664
KNN:    0.779500 (0.112294)     0.800333 (0.112294)     0.020833333333333326
NB:     0.754500 (0.105021)     0.804500 (0.105021)     0.050000000000000044
```

10.2.1 Best Models for Reduced Dataset

[Code]:
```
# If the selected model uses basic parameters, the model must be fit
simplest_best_model_name = simplest_feat_importances.nlargest(1).index[0]

if simplest_best_model_name in simplest_default_parameter: # is a vanilla model
  simplest_best_model.fit(X_train,Y_train)
```

```
Best model is LR with an accuracy of 83.35%
```

```
LogisticRegression
LogisticRegression(C=0.01, max_iter=5000, random_state=0, solver='liblinear')
```

10.2.2 Classification Report

```
              precision    recall  f1-score   support
           0       0.77      0.71      0.74        24
           1       0.82      0.86      0.84        37
    accuracy                           0.80        61
   macro avg       0.80      0.79      0.79        61
weighted avg       0.80      0.80      0.80        61
```

10.2.3 Accuracy, Prediction, Missclassification Rate

```
Accuracy: 0.8032786885245902

Precision: 0.8205128205128205

Misclassification Rate: 0.19672131147540983
```

10.2.4 Confusion Matrix, False Positive, False Negative

```
    Confusion Matrix
    [[17  7]
     [ 5 32]]

There are  7  False Positive with rate of  0.2916666666666667
There are  5  False Negative with rate of  0.13513513513513514
```

11 Comparison between Full Dataset and Reduced Dataset

11.1 Model Difference

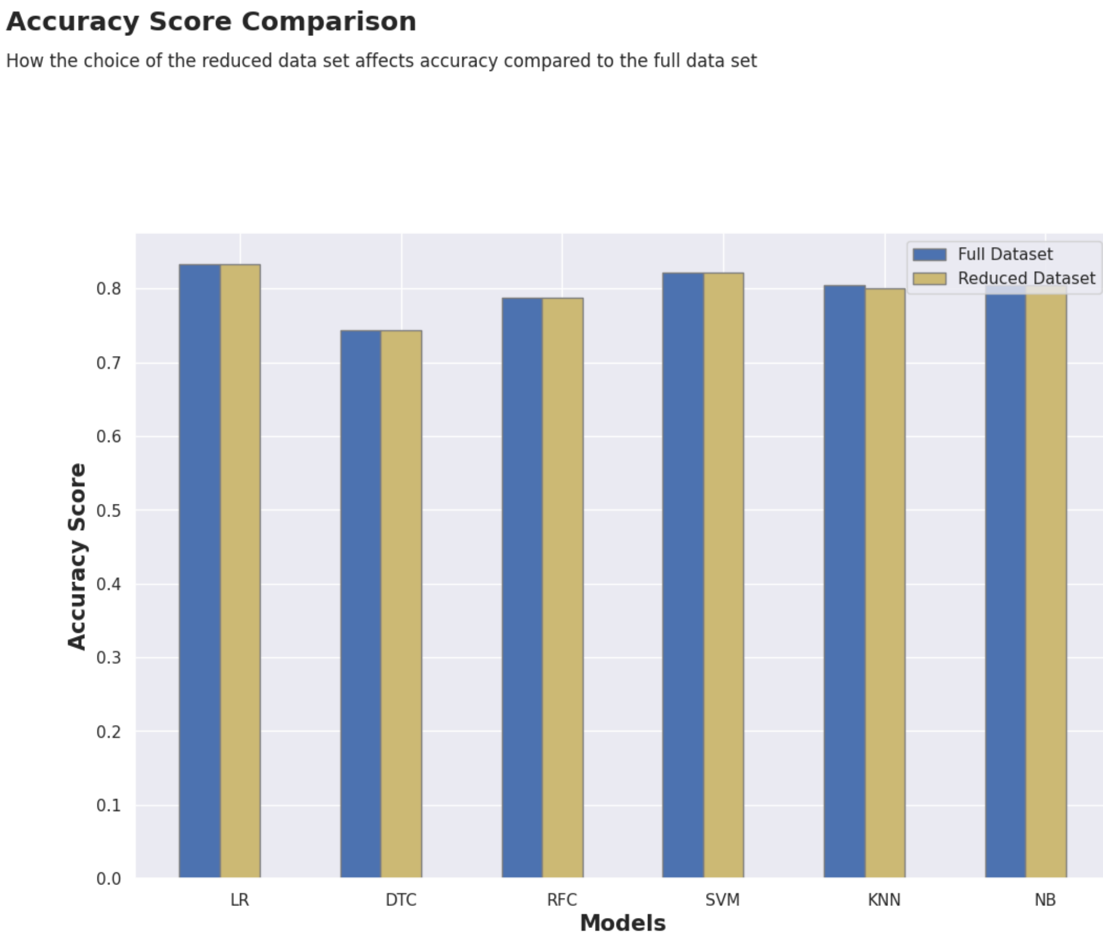
Full Dataset Model:

```
LogisticRegression(C=100, max_iter=5000, random_state=0, solver='liblinear')
```

Reduced Dataset Model:

```
LogisticRegression(C=0.01, max_iter=5000, random_state=0, solver='liblinear')
```

11.2 Accuracy Score Comparison



11.3 Classification Report Comparison

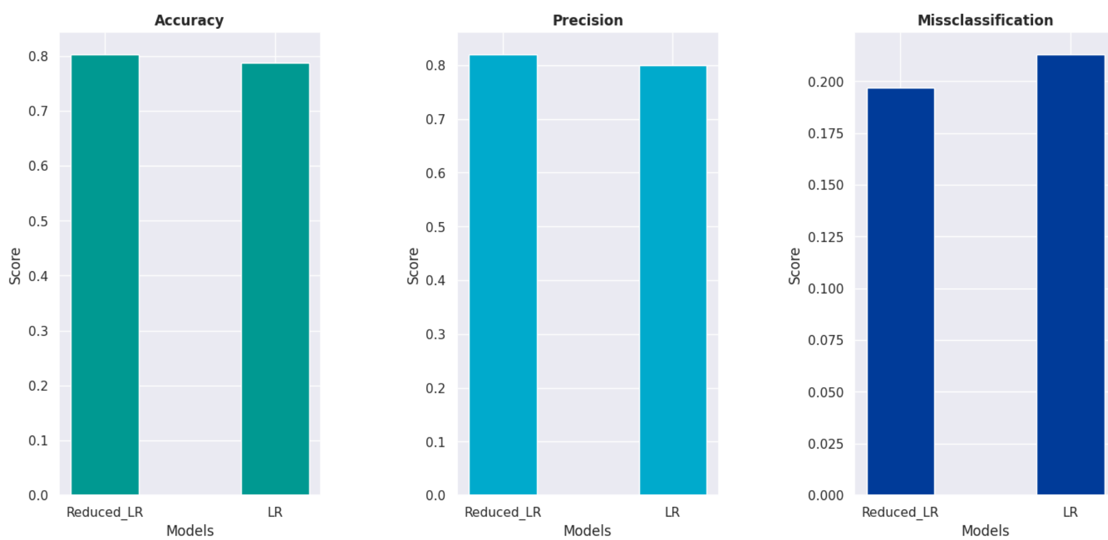
Classification Report: Reduced Dataset

	precision	recall	f1-score	support
0	0.77	0.71	0.74	24
1	0.82	0.86	0.84	37
accuracy			0.80	61
macro avg	0.80	0.79	0.79	61
weighted avg	0.80	0.80	0.80	61

Classification Report: Full Dataset

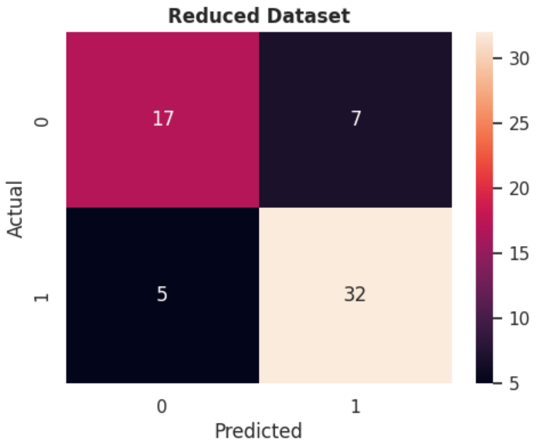
	precision	recall	f1-score	support
0	0.76	0.67	0.71	24
1	0.80	0.86	0.83	37
accuracy			0.79	61
macro avg	0.78	0.77	0.77	61
weighted avg	0.79	0.79	0.78	61

11.4 Accuracy, Precicsion, Missclassification Rate



Models	Accuracy	Precision	Missclassification
Reduced_LR:	0.8032786885245902	0.8205128205128205	0.19672131147540983
LR:	0.7868852459016393	0.8	0.21311475409836067

11.5 Confusion Matrix



11.6 False Positive, False Negative

```
Full Dataset
There are  8  False Positive with rate of  33.33 %
There are  5  False Negative with rate of  13.51 %

Reduced Dataset
There are  7  False Positive with rate of  29.17 %
There are  5  False Negative with rate of  13.51 %
```

12 Comparing the Impact of Duplicate Observations in the Machine Learning Model

This section will discuss the impact of using the initial dataset containing duplicate data.

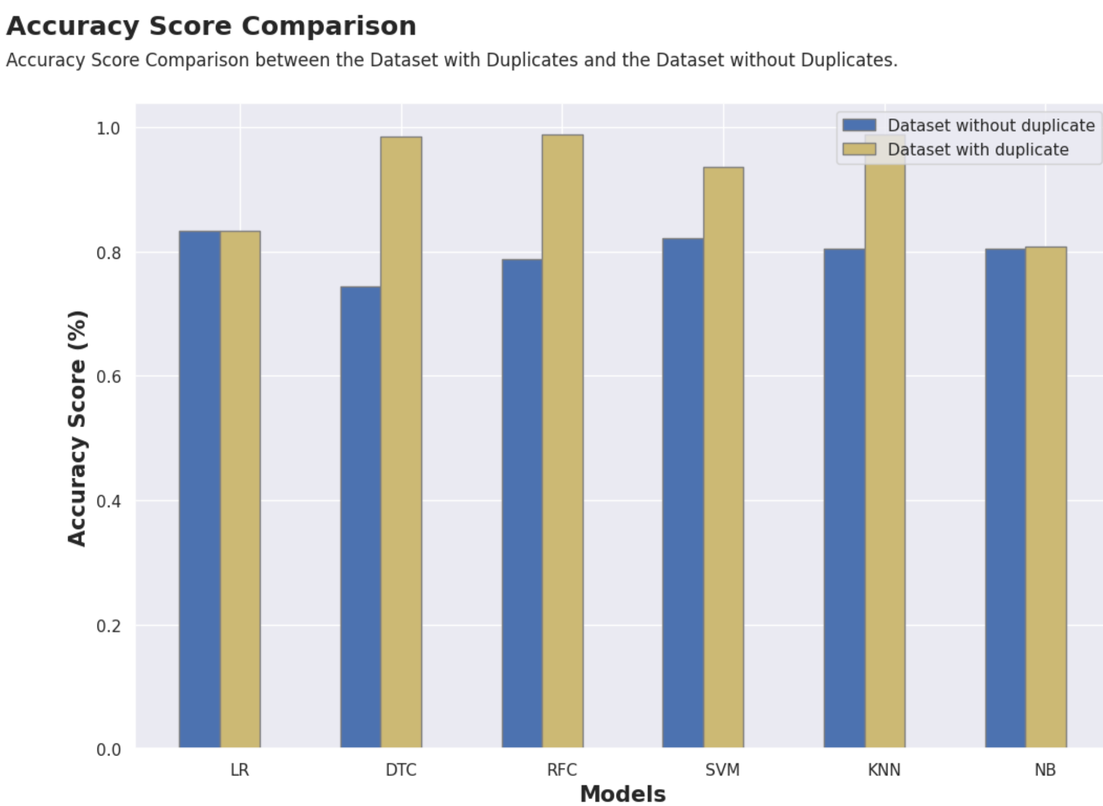
In another document (IoT Heart - M.L. with duplicated), the same Machine Learning algorithms done here have been developed, this is the outcome:

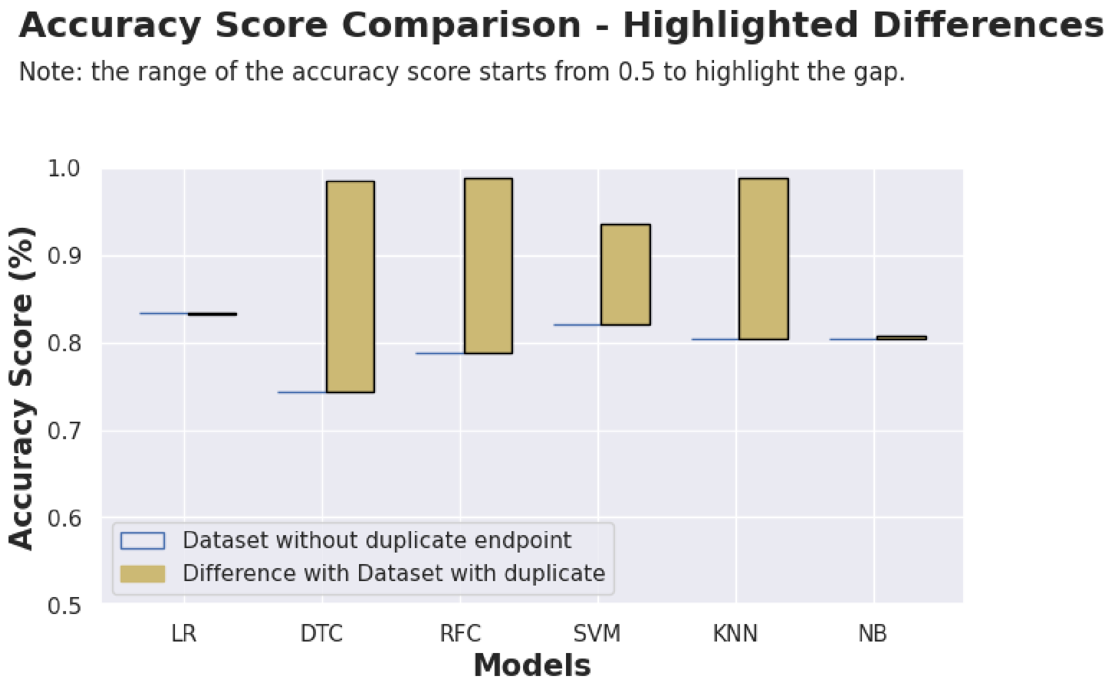
	precision	recall	f1_score	support
0	1.0	1.0	1.0	96
1	1.0	1.0	1.0	109
accuracy			1.0	205
macro_avg	1.0	1.0	1.0	205
weighted_avg	1.0	1.0	1.0	205

```
Final score:  [0.8329268292682928, 0.9853658536585366, 0.9878048780487806,
0.9365853658536587, 0.9890243902439024, 0.8073170731707318]
```

Note: The impact of removing duplicate observations is great, we went from having an accuracy of approx. 99% (with duplicates) to having an accuracy of 84% (without duplicates).

12.1 Comparing Final Score Accuracy





As can be seen, the model containing the duplicate data improves accuracy. This is because it fall into the case of overfitting: the model fits well only on this dataset, and if we were to consider different data the accuracy would decrease drastically.

This is why the model referring to the dataset without duplication is better, since in the other dataset the improvement does not correspond to reality.

13 Conclusion

- The difference found between the Reduced Dataset and the Full Dataset is slight.
- The accuracy percentage achieved by the model still has potential for improvement, although it is still good. This is because the original dataset had a limited amount of observations.
- The removal of duplicate observations had a huge impact on the model, improving the final model.

14 References

Code and complete notebook: https://colab.research.google.com/drive/16FE8Qd77dXyYvl99MaztBgt52KsmvsvQ?usp=sharing

IoT Heart - M.L. with duplicated: https://colab.research.google.com/drive/1P4SkZ7fCMX0AuaVB-FWvkln_kT7HrmTR?usp=sharing

WebMD: https://www.webmd.com/hypertension-high-blood-pressure/hypertensive-heart-disease

Image by Destin Gong on towardsdatascience.com: https://towardsdatascience.com/top-machine-learning-algorithms-for-classification-2197870ff501

Image by Gufosowa on en.wikipedia.org: https://en.wikipedia.org/wiki/Cross-validation_(statistics)#/media/File:K-fold_cross_validation_EN.svg

15 Cite

```
Dataset Cite:
Janosi,Andras, Steinbrunn,William, Pfisterer,Matthias, and Detrano,Robert.
(1988).
Heart Disease.
UCI Machine Learning Repository.
https://doi.org/10.24432/C52P4X.
```