Heart Disease Prediction

DaQuest Team

Table Of Content

Introduction

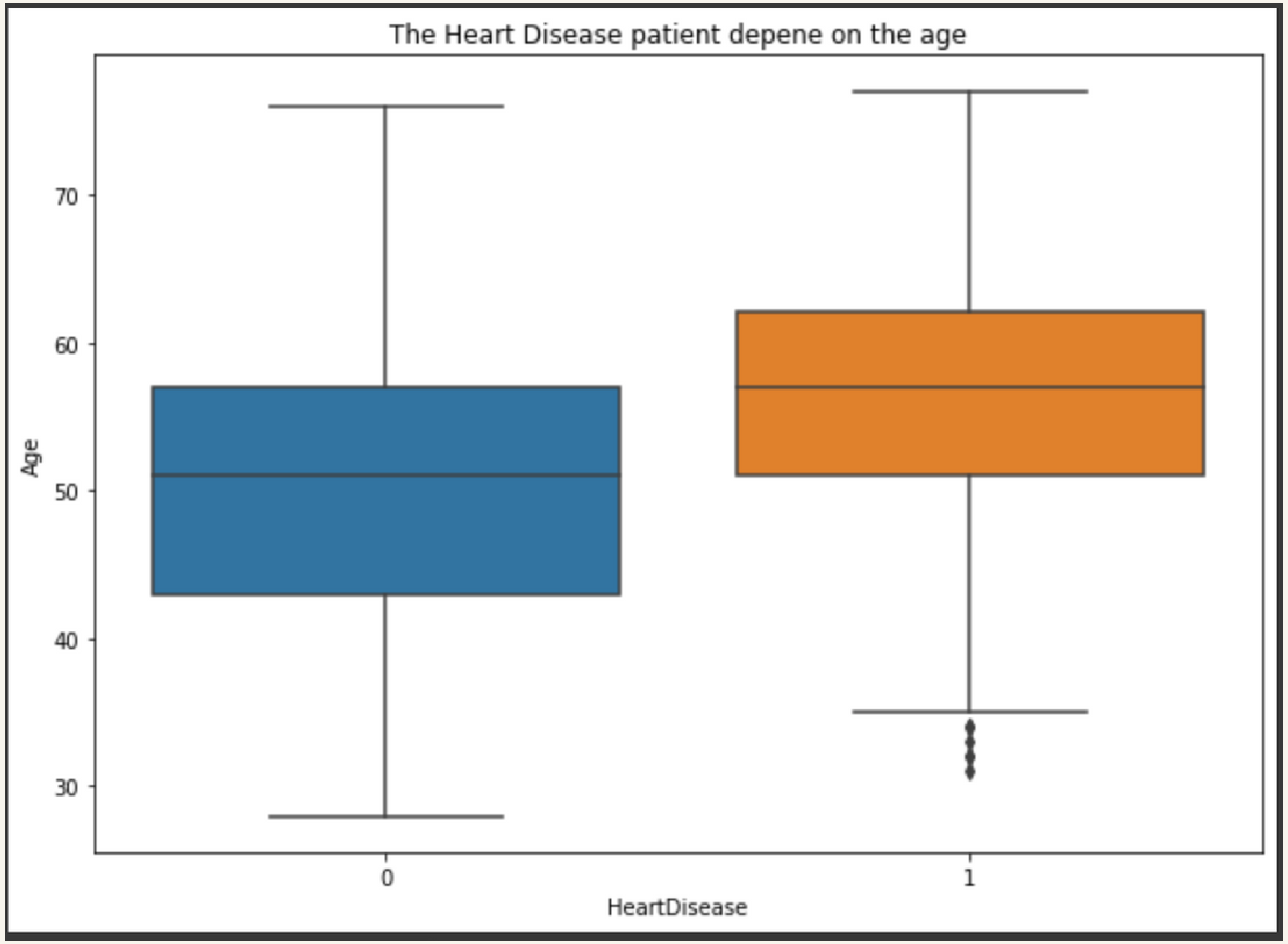
Heart failure is a common event caused by CVDs and this dataset contains 12 features that can be used to predict a possible heart disease.

Columns

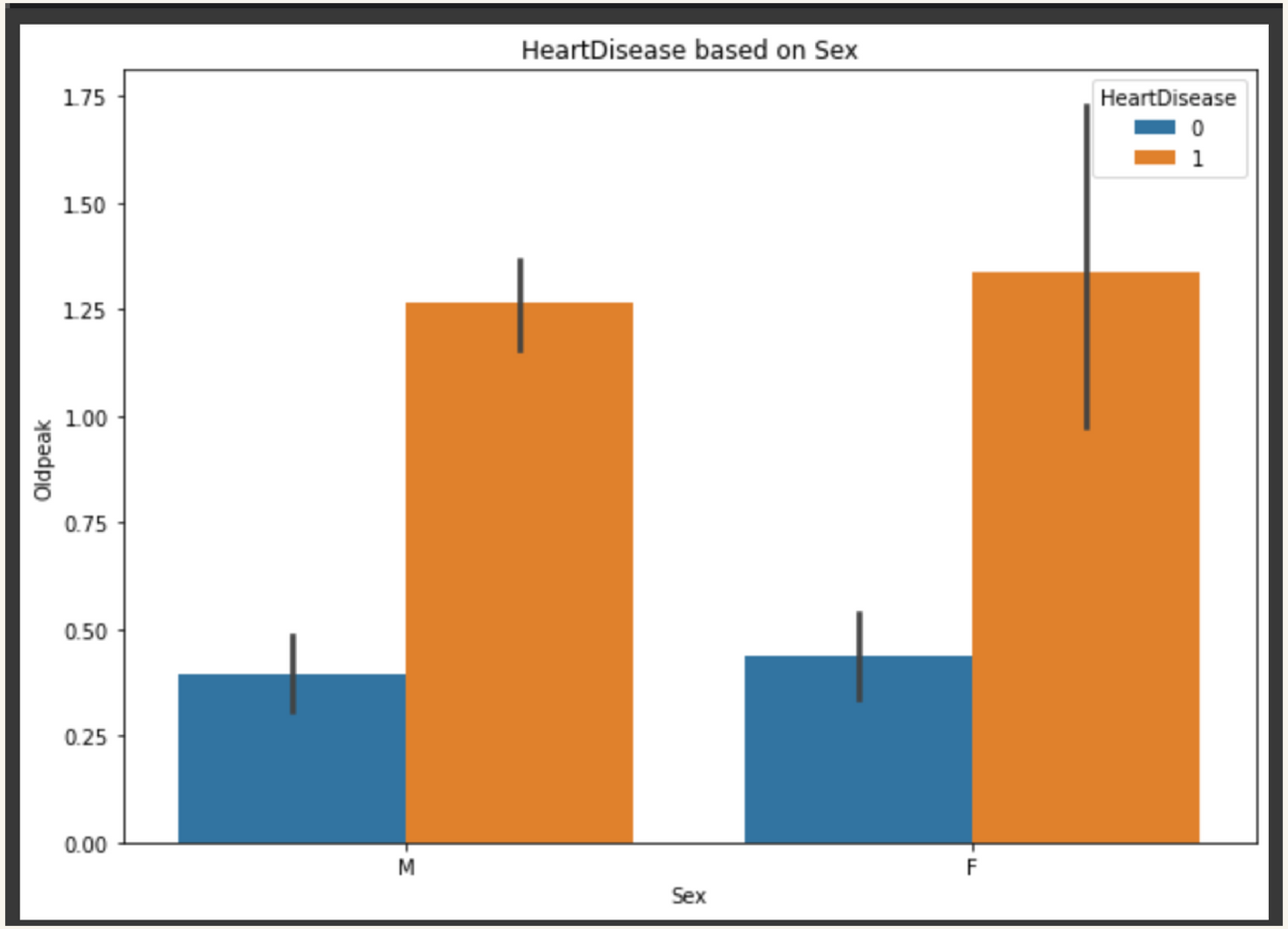
- Age
- Sex
- ChestPainType
- RestingBP
- Cholesterol
- FastingBS
- RestingECG
- MaxHR
- ExerciseAngina
- Oldpeak
- ST_Slope
- HeartDisease

Exploratory data analysis (EDA)

First Plot


The Heart Disease patient depene on the age
70
60
50
40
30
Age
0
1
HeartDisease

Second Plot



HeartDisease based on Sex
HeartDisease
0
1
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Oldpeak
M
F
Sex

Dataset Preparation

Check Null Value

```
dataset.sum().isnull()
```

```
Age               False
Sex               False
ChestPainType     False
RestingBP         False
Cholesterol       False
FastingBS         False
RestingECG        False
MaxHR             False
ExerciseAngina    False
Oldpeak           False
ST_Slope          False
HeartDisease      False
dtype: bool
```

Select Feature and Target

```
# Feature set
# The target will not be included
X = dataset.iloc[:, :-1]

# The target will be the last column
y = dataset.iloc[:, -1]
```

Encoded categorical Type

```
# Encoding categorical features
## I use the Label encoder because it's will not add more columns to the dataset and more simple to run

le = LabelEncoder()
cols = X.columns.tolist()
for column in cols:
    if X[column].dtype == 'object':
        X[column] = le.fit_transform(X[column])
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope
0	40	1	1	140	289	0	1	172	0	0.0	2
1	49	0	2	160	180	0	1	156	0	1.0	1
2	37	1	1	130	283	0	2	98	0	0.0	2
3	48	0	0	138	214	0	1	108	1	1.5	1
4	54	1	2	150	195	0	1	122	0	0.0	2

Splitting and Scaling

Split the Data Set

```
[ ] # Splitting the data to training and testing
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

Feature Scaling

```
[ ] # Feature scaling
    sc = StandardScaler()
    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)
```

Neural Networks Models

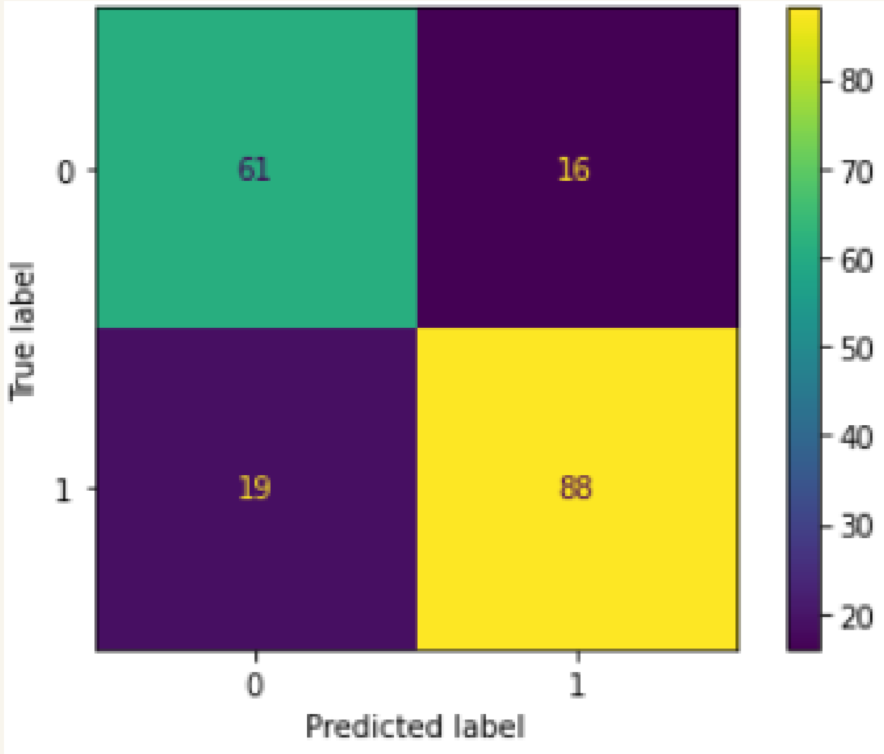
First Model :

```
#First Model
## the hidden layers for the first ANN1 model (4 layers)
ann1.add(tf.keras.layers.Dense(units=14, activation='relu'))
ann1.add(tf.keras.layers.Dense(units=14, activation='tanh'))
ann1.add(tf.keras.layers.Dense(units=12, activation='relu'))
ann1.add(tf.keras.layers.Dense(units=10, activation='relu'))
ann1.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

```
#output layer for the first ANN model
ann1.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

First Model :

	precision	recall	f1-score	support
0	0.76	0.79	0.78	77
1	0.85	0.82	0.83	107
accuracy			0.81	184
macro avg	0.80	0.81	0.81	184
weighted avg	0.81	0.81	0.81	184



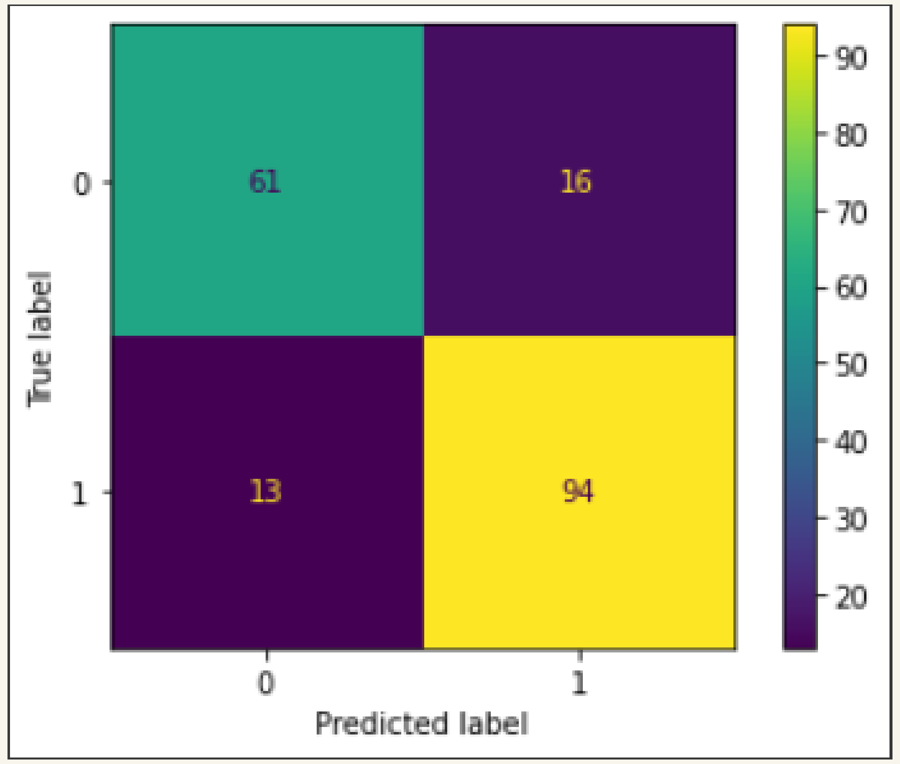
Second Model:

```
#Second Model
#Four hidden layers with TanH, RelU
ann2.add(tf.keras.layers.Dense(units=4, activation='tanh'))
ann2.add(tf.keras.layers.Dense(units=4, activation='relu'))
ann2.add(tf.keras.layers.Dense(units=4, activation='tanh'))
ann2.add(tf.keras.layers.Dense(units=4, activation='relu'))
```

```
## output layer for the Second ANN2 model
ann2.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

Second Model:

	precision	recall	f1-score	support
0	0.82	0.79	0.81	77
1	0.85	0.88	0.87	107
accuracy			0.84	184
macro avg	0.84	0.84	0.84	184
weighted avg	0.84	0.84	0.84	184



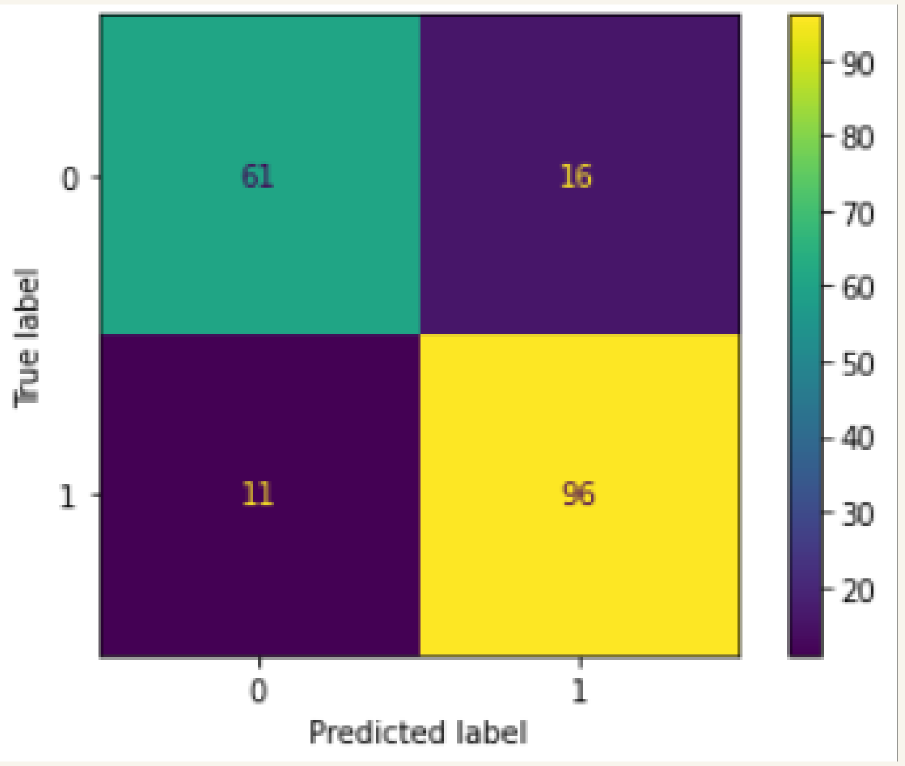
Third Model:

```
#Third Model
## the hidden layers for the Third ANN3 model (3 layers)
ann3.add(tf.keras.layers.Dense(units=12, activation='relu'))
ann3.add(tf.keras.layers.Dense(units=6, activation='relu'))
ann3.add(tf.keras.layers.Dense(units=4, activation='relu'))
```

```
## output layer for the Third ANN3 model
ann3.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

Third Model:

	precision	recall	f1-score	support
0	0.85	0.79	0.82	77
1	0.86	0.90	0.88	107
accuracy			0.85	184
macro avg	0.85	0.84	0.85	184
weighted avg	0.85	0.85	0.85	184



Fourth Model :

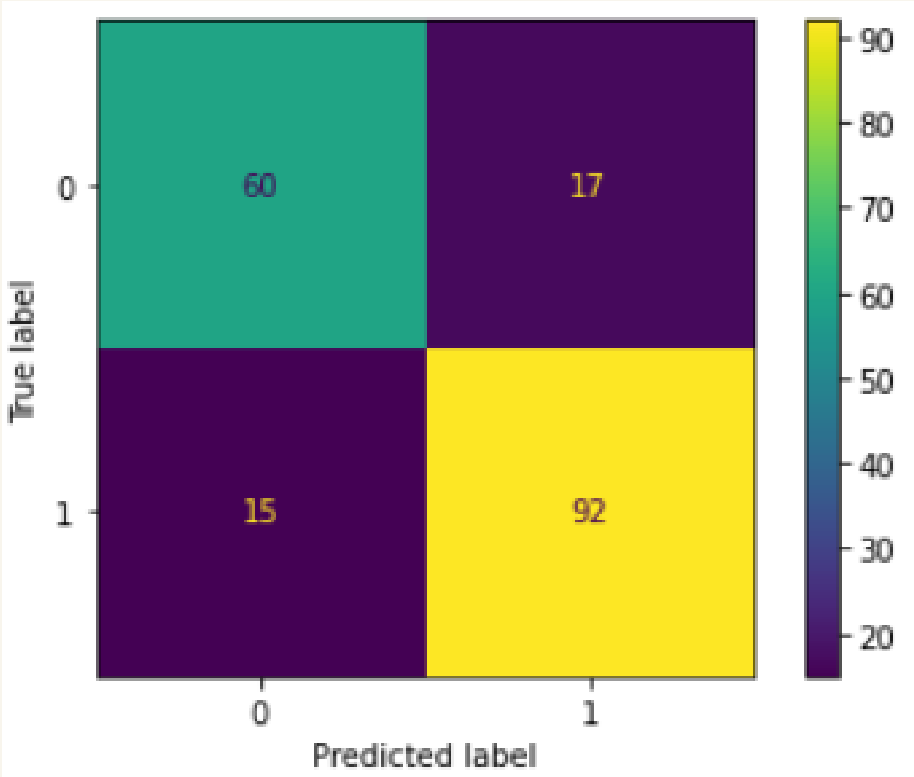
```
#Fourth Model
## the hidden layers for the Fourth ANN4 model (3 layers)

ann4.add(tf.keras.layers.Dense(units=6, activation='relu'))
ann4.add(tf.keras.layers.Dense(units=4, activation='relu'))
ann4.add(tf.keras.layers.Dense(units=4, activation='relu'))
```

```
## output layer for the Fourth ANN4 model
ann4.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

Fourth Model :

	precision	recall	f1-score	support
0	0.80	0.78	0.79	77
1	0.84	0.86	0.85	107
accuracy			0.83	184
macro avg	0.82	0.82	0.82	184
weighted avg	0.83	0.83	0.83	184

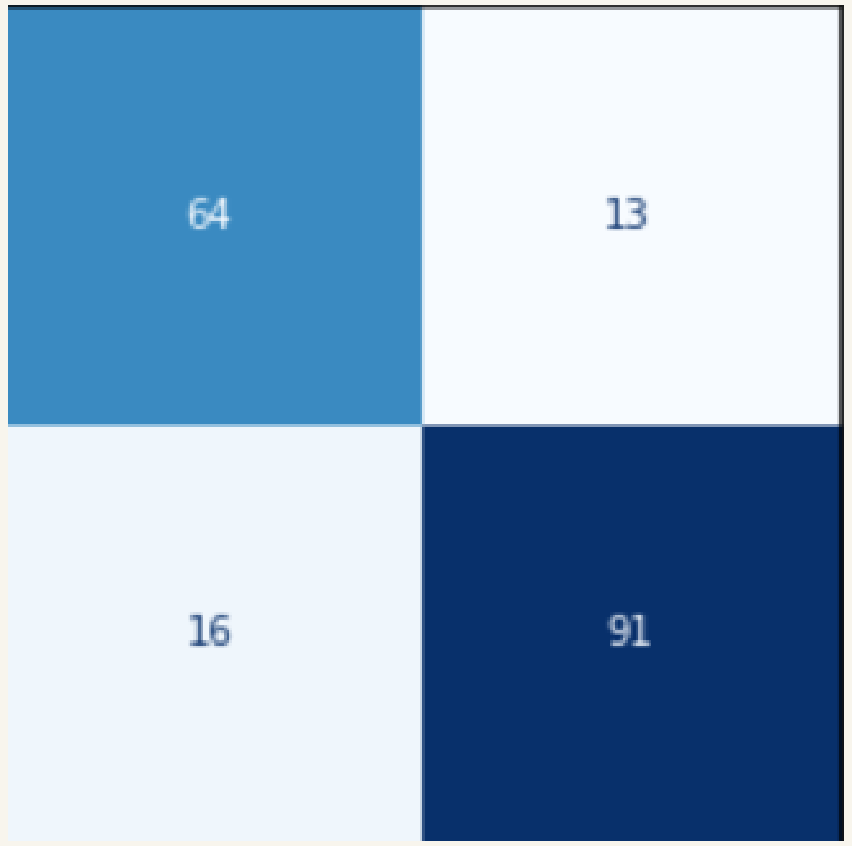


Classification Models

GridSearch

DecisionTreeClassifier

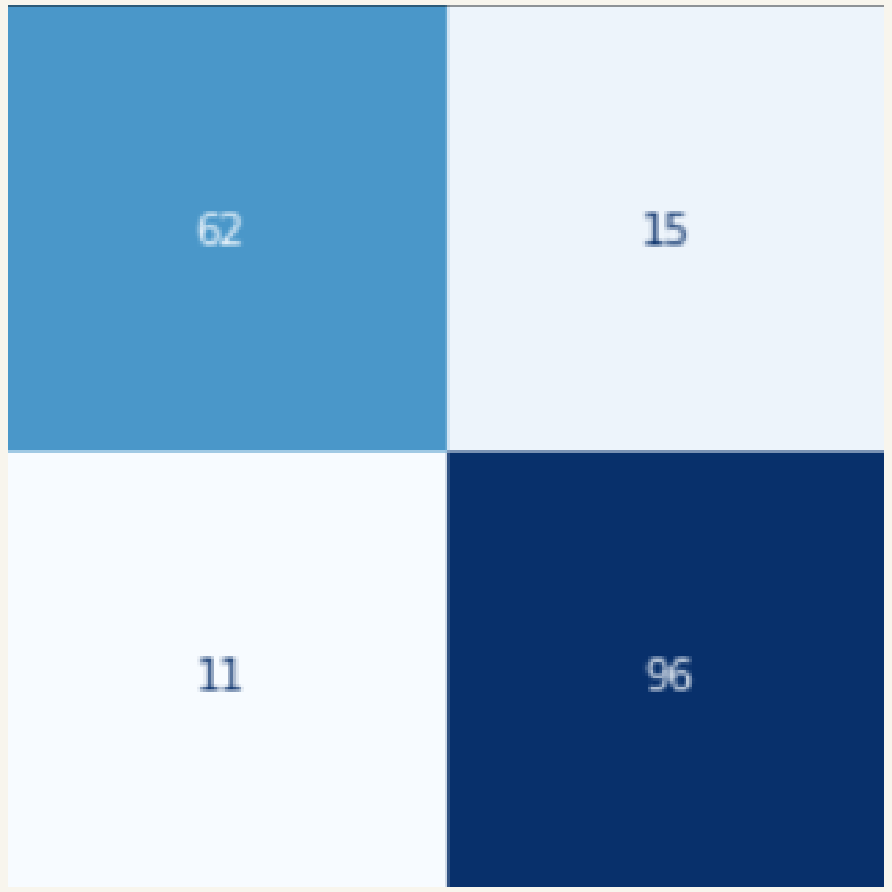
	precision	recall	f1-score	support
0	0.80	0.83	0.82	77
1	0.88	0.85	0.86	107
accuracy			0.84	184
macro avg	0.84	0.84	0.84	184
weighted avg	0.84	0.84	0.84	184



GridSearch

RandomForestClassifier

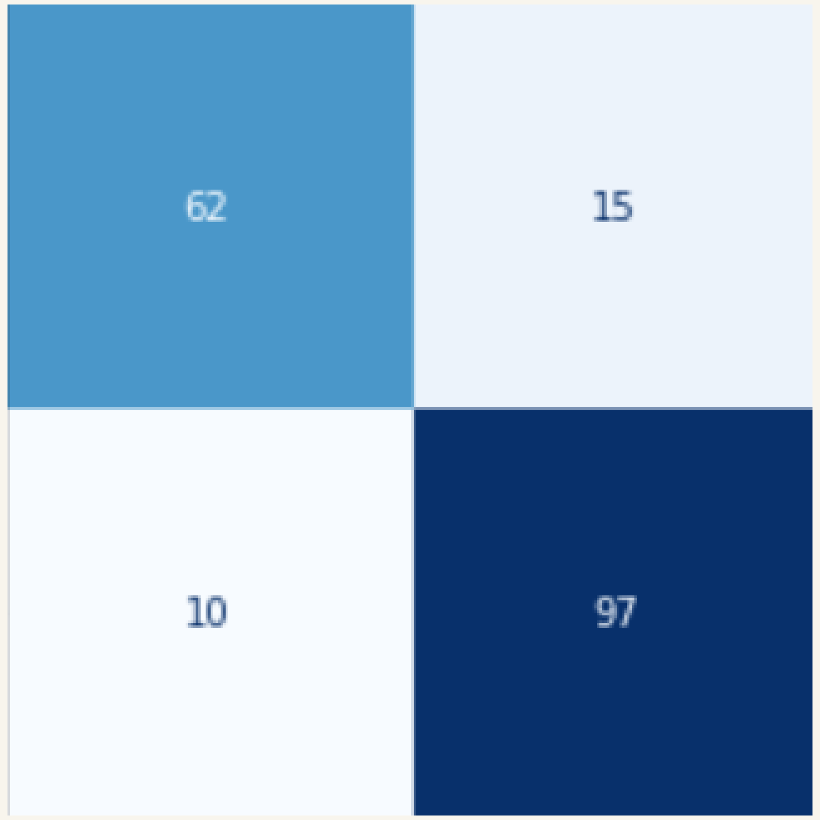
	precision	recall	f1-score	support
0	0.85	0.81	0.83	77
1	0.86	0.90	0.88	107
accuracy			0.86	184
macro avg	0.86	0.85	0.85	184
weighted avg	0.86	0.86	0.86	184



GridSearch

Support Vector Classifier

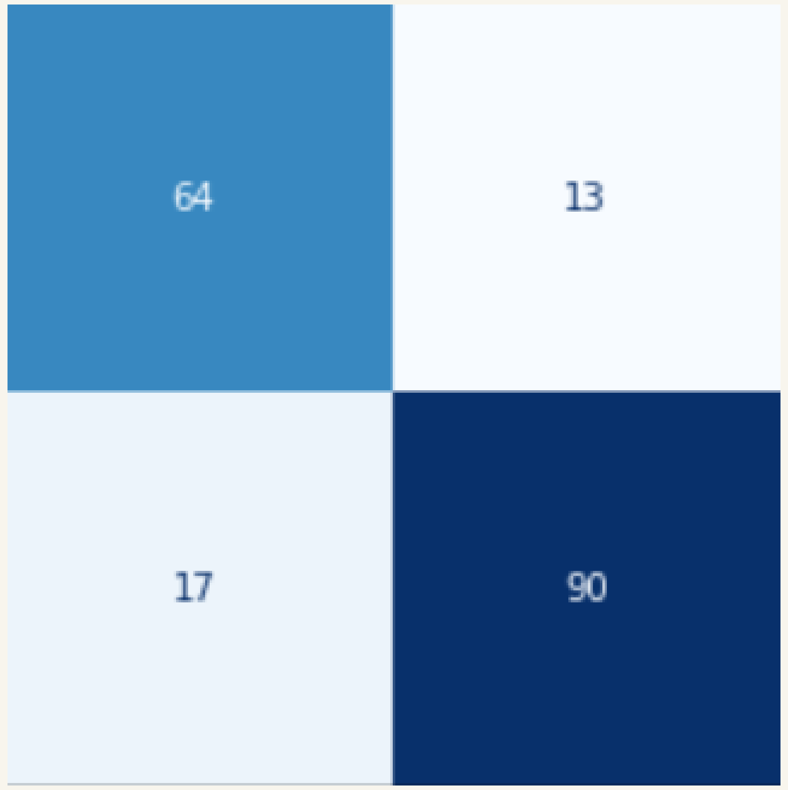
	precision	recall	f1-score	support
0	0.86	0.81	0.83	77
1	0.87	0.91	0.89	107
accuracy			0.86	184
macro avg	0.86	0.86	0.86	184
weighted avg	0.86	0.86	0.86	184



GridSearch

KNeighborsClassifier

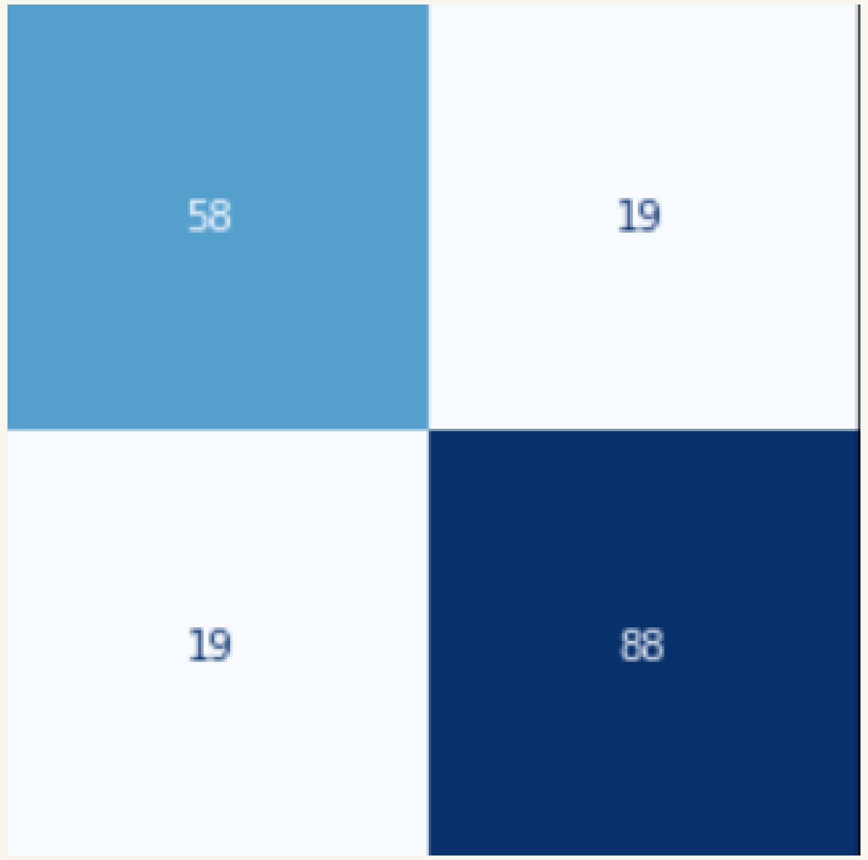
	precision	recall	f1-score	support
0	0.79	0.83	0.81	77
1	0.87	0.84	0.86	107
accuracy			0.84	184
macro avg	0.83	0.84	0.83	184
weighted avg	0.84	0.84	0.84	184



GridSearch

XGBClassifier

	precision	recall	f1-score	support
0	0.75	0.75	0.75	77
1	0.82	0.82	0.82	107
accuracy			0.79	184
macro avg	0.79	0.79	0.79	184
weighted avg	0.79	0.79	0.79	184



Model List

using the GridSearch best parameter

```
models_list = [LogisticRegression(C = 100.0, penalty = 'l2', solver = 'lbfgs'),
               DecisionTreeClassifier(max_leaf_nodes = 28, min_samples_split = 4),
               RandomForestClassifier(max_depth= None, n_estimators = 100),
               SVC(C = 1000, gamma = 0.0001, kernel= 'rbf'),
               KNeighborsClassifier(n_neighbors = 17, weights = 'uniform'),
               XGBClassifier(colsample_bytree = 0.7, learning_rate = 0.05, max_depth = 6, min_child_weight = 11,
                             missing = -999, n_estimators = 5, nthread = 4, objective ='binary:logistic', seed = 1337,
                             silent = 1, subsample = 0.8)
               ]
```

Model Evaluation

```
df_perf_metrics = pd.DataFrame(columns=[
    'Model', 'Accuracy_Training_Set', 'Accuracy_Test_Set', 'Precision',
    'Recall', 'f1_score'
])
models_trained_list = []

def get_perf_metrics(model, i):
    # model name
    model_name = type(model).__name__
    print("Training {} model...".format(model_name))
    # Fitting of model
    model.fit(X_train, y_train)
    print("Completed {} model training.".format(model_name))
    # Predictions
    y_pred = model.predict(X_test)
    # Add to ith row of dataframe - metrics

    df_perf_metrics.loc[i] = [
        model_name,
        model.score(X_train, y_train),
        model.score(X_test, y_test),
        precision_score(y_test, y_pred),
        recall_score(y_test, y_pred),
        f1_score(y_test, y_pred),
    ]

    print("Completed {} model's performance assessment.".format(model_name))
```

ANN Evaluation

```
nn_list = [history1, history2, history3, history4]
pred_list = [y_pred1, y_pred2, y_pred3, y_pred4]

def get_nn_metrics(history_, n):
    train_acc = history_.history['accuracy'][-1]
    df_perf_metrics.loc[len(df_perf_metrics.index)] = [
        'ANN '+ str(n+1),
        train_acc,
        accuracy_score(y_test, pred_list[n]),
        precision_score(y_test, pred_list[n]),
        recall_score(y_test, pred_list[n]),
        f1_score(y_test, pred_list[n])
    ]

for n, history in enumerate(nn_list):
    get_nn_metrics(history, n)
```

Comparison

```
df_perf_metrics
```

	Model	Accuracy_Training_Set	Accuracy_Test_Set	Precision	Recall	f1_score
0	LogisticRegression	0.859673	0.831522	0.839286	0.878505	0.858447
1	DecisionTreeClassifier	0.918256	0.804348	0.831776	0.831776	0.831776
2	RandomForestClassifier	1.000000	0.853261	0.850877	0.906542	0.877828
3	SVC	0.869210	0.831522	0.839286	0.878505	0.858447
4	KNeighborsClassifier	0.870572	0.836957	0.881188	0.831776	0.855769
5	XGBClassifier	0.851499	0.793478	0.822430	0.822430	0.822430
6	ANN 1	0.901907	0.809783	0.846154	0.822430	0.834123
7	ANN 2	0.895095	0.842391	0.854545	0.878505	0.866359
8	ANN 3	0.933242	0.853261	0.857143	0.897196	0.876712
9	ANN 4	0.869210	0.826087	0.844037	0.859813	0.851852

Thank You For Listening

Any Question?