Heart Failure Dataset

This dataset was sourced from Kaggle.

This dataset has 13 columns and 304 rows.

The name of columns are as follows with their metrics and absolute correlation with target variable-

Metrics	Mean	Standard Deviation	Absolute Correlation
sex	54.36634	9.082101	0.225438716
cp	0.683168	0.466011	0.280936576
trestbps	0.966997	1.032052	0.433798262
chol	131.6238	17.53814	0.144931128
fbs	246.264	51.83075	0.085239105
restecg	0.148515	0.356198	-0.02804576
thalach	0.528053	0.52586	0.137229503
exang	149.6469	22.90516	0.421740934
oldpeak	0.326733	0.469794	0.436757083
slope	1.039604	1.161075	0.430696002
ca	1.39934	0.616226	0.345877078
thal	0.729373	1.022606	0.391723992
target	2.313531	0.612277	0.344029268

We are going to predict the target column from all the other columns in the dataset.

The target variable has two values of 0 and 1 where 0 means no heart failure and 1 means their heart was failed.

The value counts of 0 and 1 are as follows-:

1 : 165
0 : 138

These are the values counts of the rest of the varibles-:

age 41
sex 2
cp 4
trestbps 49
chol 152
fbs 2
restecg 3
thalach 91
exang 2
oldpeak 40
slope 3
ca 5

thal 4
target 2

The categorical variables which had less then 6 values were encoded using pd.get_dummies().

After that dataset had 303 rows × 21 columns.

The pre-processing ends here.

The dataset was then split into train and test sets using stratifies split The code used was as follows-:

```
feature_cols = [x for x in data.columns if x!='target']

from sklearn.model_selection import StratifiedShuffleSplit


sss = StratifiedShuffleSplit(n_splits=1, test_size=100)

train_idx, test_idx = next(sss.split(data[feature_cols], data['target']))

X_train = data.loc[train_idx, feature_cols]

X_test = data.loc[test_idx, feature_cols]

Y_train = data.loc[train_idx, 'target']

Y_test = data.loc[test_idx, 'target']
```

The first model was apllied was LinearSVM which was applied using this code-:

```
from sklearn.svm import LinearSVC


Lsvc = LinearSVC()

Lsvc.fit(X_train, Y_train)

y_pred = Lsvc.predict(X_test)


from sklearn.metrics import accuracy_score, f1_score, recall_score, classification_report, confusion_matrix

print(classification_report(Y_test, y_pred))

print(f'Recall:\n {recall_score(Y_test, y_pred)}')

print(f'Accuracy_score:\n{accuracy_score(Y_test, y_pred)}')

print(f'f1_score:\n{f1_score(Y_test, y_pred)}')
```

The metrics from first model was as follows-:

```
precision    recall  f1-score   support

          0       0.92      0.26      0.41        46
          1       0.61      0.98      0.75        54

avg / total       0.75      0.65      0.59       100

Recall:
 0.9814814814814815
Accuracy_score:
0.65
f1_score:
0.75177304964539
```

This model did not perform well but the plot is as follows

Series 1 is true value and Series 2 is predicted values

The next model was Gaussian SVM with Grid SearchCV-:

The code was used was as follows-:

```
param_grid = {'gamma':[0.001,0.01,0.1,0.5,1,2,10],
          'C':[0.01,0.1,1,10]}
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
GS_SVC = GridSearchCV(SVC(kernel='rbf'),
                 param_grid=param_grid,
```

```
                     n_jobs=-1,
                    scoring='accuracy')

GS_SVC.fit(X_train, Y_train)
GS_SVC.best_estimator_

y_pred = GS_SVC.predict(X_test)

print(classification_report(Y_test, y_pred))
print(f'Recall:\n {recall_score(Y_test, y_pred)}')
print(f'Accuracy_score:\n{accuracy_score(Y_test, y_pred)}')
print(f'f1_score:\n{f1_score(Y_test, y_pred)}')

gasvm = pd.DataFrame(data=[Y_test, y_pred]).T
gasvm['Unnamed 0'] = linsvm['Unnamed 0'].fillna(0)
gasvm
gasvm.to_csv('gasvm.csv')
```

The metrics were as follows-:

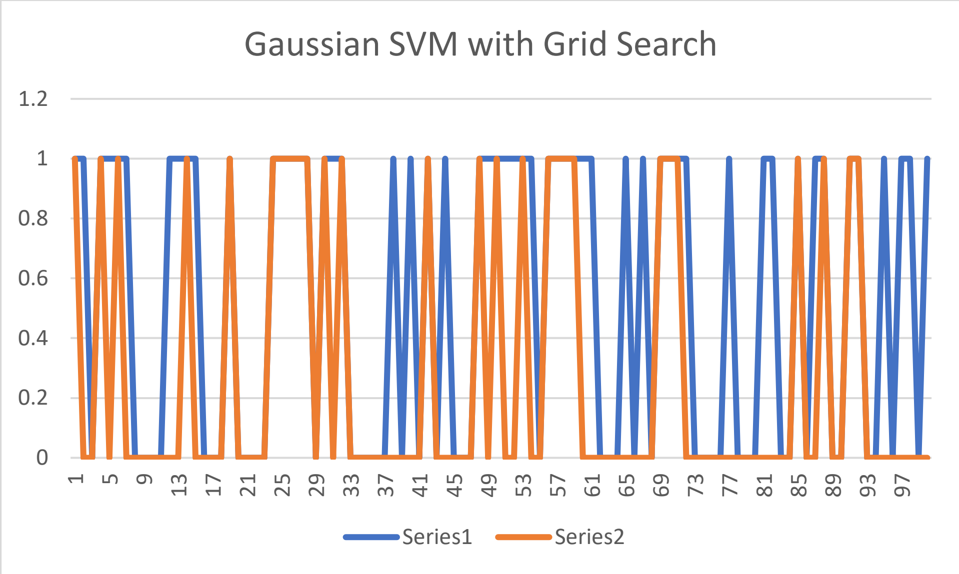
```
      precision    recall  f1-score   support

          0      0.66      0.59      0.62       46
          1      0.68      0.74      0.71       54

avg / total      0.67      0.67      0.67      100

Recall:
 0.7407407407407407
Accuracy_score:
0.67
f1_score:
0.7079646017699114
```

The plot is as follows-:



This model was performed worse than LinearSVM

The Final model was descisionTrees with grid searchCV

The code used was as follows-:

```
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier()

dt = dt.fit(X_train, Y_train)


param_grid = {'max_depth':range(1, dt.tree_.max_depth+1,2),

        'max_features':range(1, len(dt.feature_importances_)+1)}

GR_DT = GridSearchCV(DecisionTreeClassifier(),

            param_grid = param_grid,

            scoring='accuracy',

            n_jobs=-1)

GR_DT = GR_DT.fit(X_train, Y_train)

y_pred= GR_DT.predict(X_test)

print(classification_report(Y_test, y_pred))

print(f'Recall:\n {recall_score(Y_test, y_pred)}')

print(f'Accuracy_score:\n{accuracy_score(Y_test, y_pred)}')

print(f'f1_score:\n{f1_score(Y_test, y_pred)}')
```

dtsvm = pd.DataFrame(data=[Y_test, y_pred]).T

dtsvm['Unnamed 0'] = linsvm['Unnamed 0'].fillna(0)

dtsvm

dtsvm.to_csv('dtsvm.csv')

The error metrics were as follows for this model-:

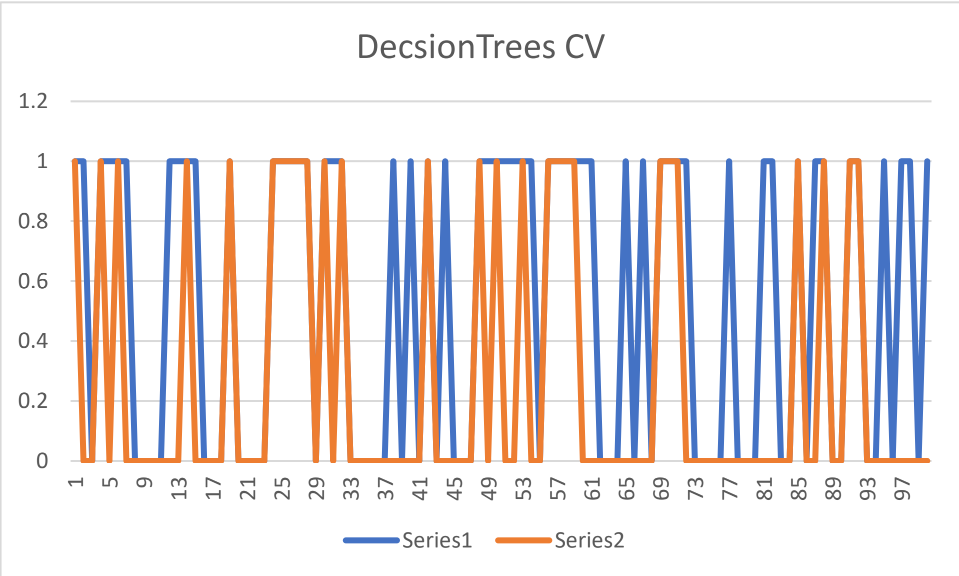
```
             precision    recall  f1-score   support

          0       0.74      0.61      0.67        46
          1       0.71      0.81      0.76        54

avg / total       0.72      0.72      0.72       100

Recall:
 0.8148148148148148
Accuracy_score:
0.72
f1_score:
0.7586206896551724
```

This model also performed bad than LinearSVM. The plot is as follows-:



We conclude that LinearSVM was the best model for our dataset

The dataset can be improved by adding more rows and providing a little bit more insight on the health of the patient.