Final Project ML: Heart Failure Prediction

IBM Machine Learning Professional Certificate

Supervised Machine Learning: Classification

Shaula Marquez

Contents

- Data Description
- Main Objective
- Data Analysis
- Classification models
- Analysis and Findings
- Conclusion

Data Description

Supervised Machine Learning: Classification

Heart Failure Analysis Prediction

Introduction

Cardiovascular diseases (CVDs) are the number 1 cause of death globally, taking an estimated 17.9 million lives each year, which accounts for 31% of all deaths worldwide.

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

Most cardiovascular diseases can be prevented by addressing behavioral risk factors such as tobacco use, unhealthy diet and obesity, physical inactivity and harmful use of alcohol using population-wide strategies.

People with cardiovascular disease or who are at high cardiovascular risk (due to the presence of one or more risk factors such as hypertension, diabetes, hyperlipidemia or already established disease) need early detection and management wherein a machine learning model can be of great help.

Dataset: https://www.kaggle.com/datasets/andrewmvd/heart-failure-clinical-data

Dataset Description

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	smoking	time	DEATH_EVENT
0	75.0	0	582	0	20	1	265000.00	1.9	130	1	0	4	1
1	55.0	0	7861	0	38	0	263358.03	1.1	136	1	0	6	1
2	65.0	0	146	0	20	0	162000.00	1.3	129	1	1	7	1
3	50.0	1	111	0	20	0	210000.00	1.9	137	1	0	7	1
4	65.0	1	160	1	20	0	327000.00	2.7	116	0	0	8	1

- **Age:** Age of the patient
- **Anemia:** Hemoglobin level of patient (Boolean)
- **Creatinine Phosphokinase:** Level of the CPK enzyme in the blood (mcg/L)
- **Diabetes:** If the patient has diabetes (Boolean)
- **Ejection fraction:** Percentage of blood leaving the heart at each contraction
- **High blood pressure:** If the patient has hypertension (Boolean)
- **Platelets:** Platelet count of blood (kilo platelets/mL)

Dataset Description

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	smoking	time	DEATH_EVENT
0	75.0	0	582	0	20	1	265000.00	1.9	130	1	0	4	1
1	55.0	0	7861	0	38	0	263358.03	1.1	136	1	0	6	1
2	65.0	0	146	0	20	0	162000.00	1.3	129	1	1	7	1
3	50.0	1	111	0	20	0	210000.00	1.9	137	1	0	7	1
4	65.0	1	160	1	20	0	327000.00	2.7	116	0	0	8	1

- **Serum creatinine:** Level of serum creatinine in the blood (mg/dL)
- **Serum sodium:** Level of serum sodium in the blood (mEq/L)
- **Sex:** Sex of the patient
- **Smoking:** If the patient smokes or not (Boolean)
- **Time:** Follow-up period (days)
- **DEATH_EVENT:** If the patient deceased during the follow-up period (Boolean)
- **Attributes having Boolean values:**

  0 = Negative (No); 1 = Positive (Yes)

Dataset Description: Statistical

	count	mean	std	min	25%	50%	75%	max
age	299.0	60.833893	11.894809	40.0	51.0	60.0	70.0	95.0
anaemia	299.0	0.431438	0.496107	0.0	0.0	0.0	1.0	1.0
creatinine_phosphokinase	299.0	581.839465	970.287881	23.0	116.5	250.0	582.0	7861.0
diabetes	299.0	0.418060	0.494067	0.0	0.0	0.0	1.0	1.0
ejection_fraction	299.0	38.083612	11.834841	14.0	30.0	38.0	45.0	80.0
high_blood_pressure	299.0	0.351171	0.478136	0.0	0.0	0.0	1.0	1.0
platelets	299.0	263358.029264	97804.236869	25100.0	212500.0	262000.0	303500.0	850000.0
serum_creatinine	299.0	1.393880	1.034510	0.5	0.9	1.1	1.4	9.4
serum_sodium	299.0	136.625418	4.412477	113.0	134.0	137.0	140.0	148.0
sex	299.0	0.648829	0.478136	0.0	0.0	1.0	1.0	1.0
smoking	299.0	0.321070	0.467670	0.0	0.0	0.0	1.0	1.0
time	299.0	130.260870	77.614208	4.0	73.0	115.0	203.0	285.0
DEATH_EVENT	299.0	0.321070	0.467670	0.0	0.0	0.0	1.0	1.0

Dataset Description: Missing Values

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

✓ 0.0s

```
age                         0
anaemia                     0
creatinine_phosphokinase    0
diabetes                    0
ejection_fraction           0
high_blood_pressure         0
platelets                   0
serum_creatinine            0
serum_sodium                0
sex                         0
smoking                     0
time                        0
DEATH_EVENT                 0
dtype: int64
```

There are no missing values.

Main Objective of the Analysis:

The primary objective of this analysis is to develop a predictive model that accurately identifies individuals at risk of heart failure based on key clinical and demographic features. By analyzing the correlation between different factors, we aim to determine the most influential predictors of heart failure.

To achieve this, we employ various machine learning classification models, leveraging techniques such as GridSearch, ML pipelines, and hyperparameter tuning to optimize performance. The goal is to select the most effective model in terms of accuracy while assessing the limitations of each approach. Ultimately, this analysis seeks to contribute to early detection and improved clinical decision-making for heart failure patients.

Data Analysis

Supervised Machine Learning: Classification

Heart Failure Analysis Prediction

Data Analysis: Identifying Features

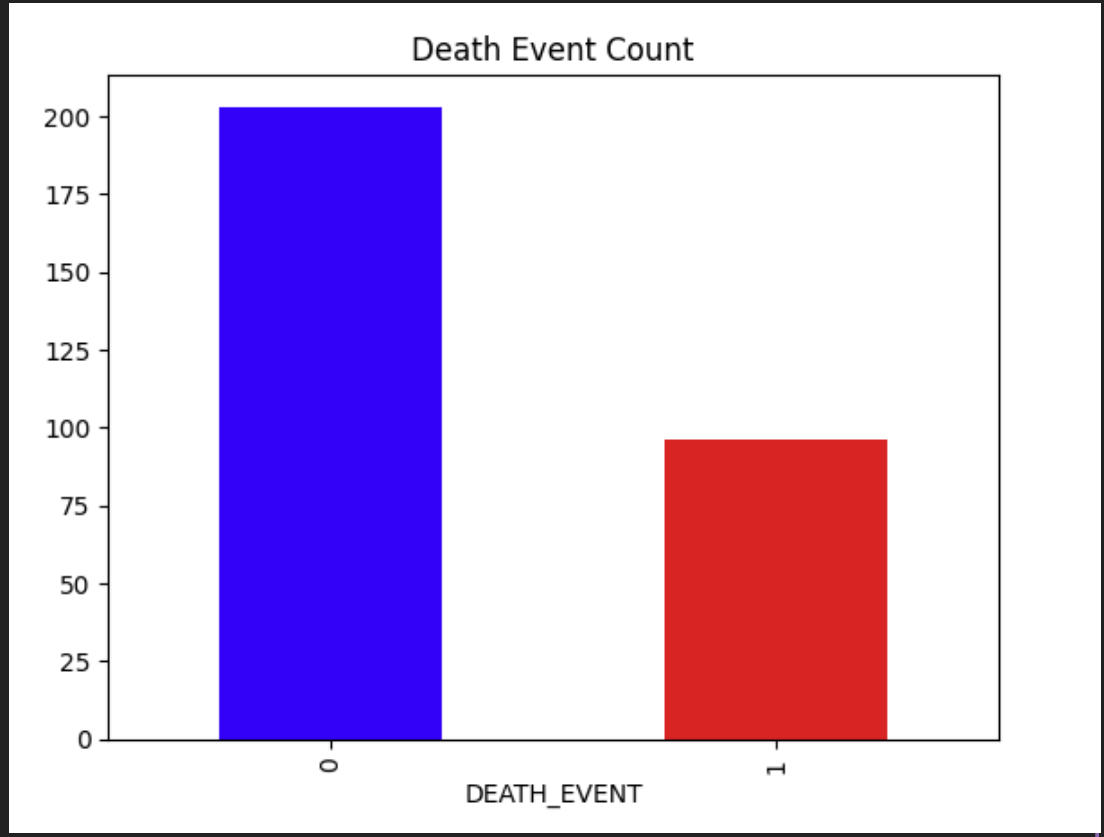
We Identified the Categorical Features and Continous Features.

```
==============================
Categorical Features : ['anaemia', 'diabetes', 'high_blood_pressure', 'sex', 'smoking', 'DEATH_EVENT']
Continous Features : ['age', 'creatinine_phosphokinase', 'ejection_fraction', 'platelets', 'serum_creatinine', 'serum_sodium', 'time']
```

Data Analysis: Death Event Count

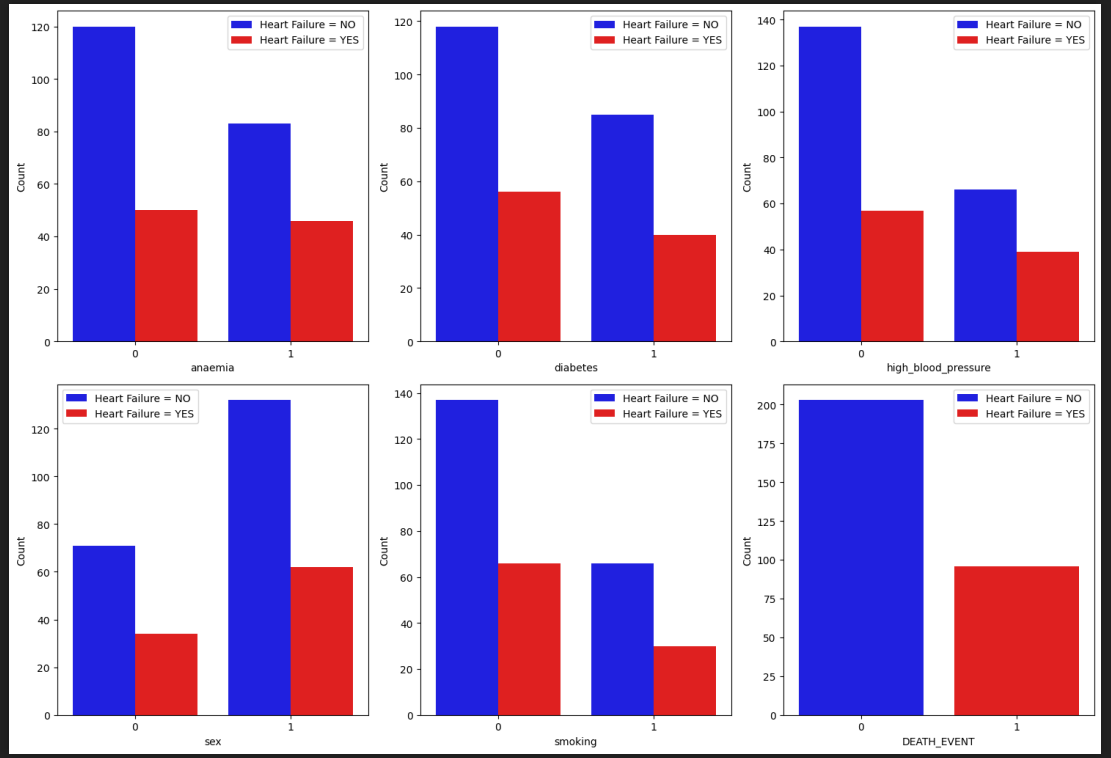
We have 96 people who died due to Heart Failure and 203 people who survived.

```
DEATH_EVENT
0    203
1     96
Name: count, dtype: int64
```



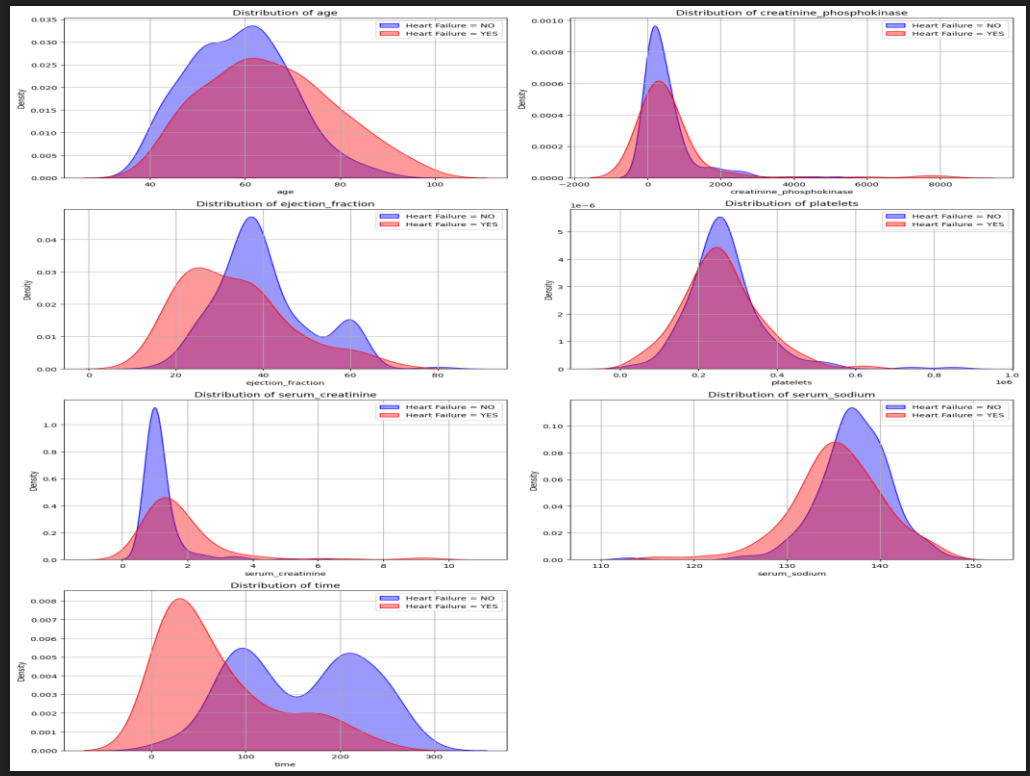
Data Analysis: Categorical Features

- High blood pressure, anemia, diabetes, and smoking seem to be potential risk factors for heart failure.
- Males appear to be more affected than females in this dataset.



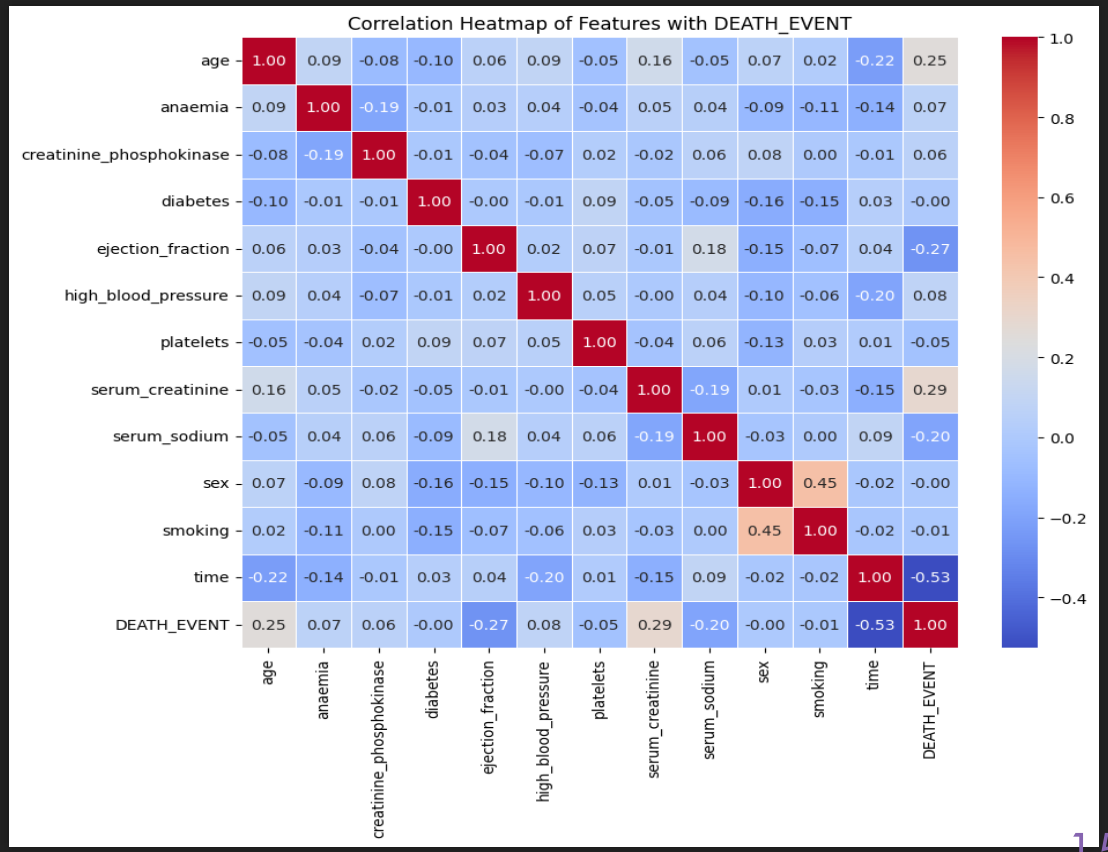
Data Analysis: Continous Features

- Age, ejection fraction, serum creatinine, and serum sodium appear to be key differentiators between heart failure and non-heart failure patients.
- Platelet count does not show a clear separation, meaning it might not be a strong predictor.
- Higher creatinine and lower sodium levels suggest potential kidney involvement in heart failure.
- Time differences indicate that heart failure patients might have shorter survival or follow-up durations.



Data Analysis: Correlation

- Age, ejection fraction, serum creatinine, and serum sodium appear to be key differentiators between heart failure and non-heart failure patients.
- Platelet count does not show a clear separation, meaning it might not be a strong predictor.
- Higher creatinine and lower sodium levels suggest potential kidney involvement in heart failure.
- Time differences indicate that heart failure patients might have shorter survival or follow-up durations.



Data Analysis: Feature Engineering

	age	creatinine_phosphokinase	ejection_fraction	platelets	serum_creatinine	serum_sodium	time	DEATH_EVENT	anaemia_0	anaemia_1	diabetes_0	diabetes_1	high_blood_pressure_0
0	1.192945	0.000166	-1.530560	1.681648e-02	0.490057	-1.504036	-1.629502	1	True	False	True	False	False
1	-0.491279	7.514640	-0.007077	7.535660e-09	-0.284552	-0.141976	-1.603691	1	True	False	True	False	True
2	0.350833	-0.449939	-1.530560	-1.038073e+00	-0.090900	-1.731046	-1.590785	1	True	False	True	False	True
3	-0.912335	-0.486071	-1.530560	-5.464741e-01	0.490057	0.085034	-1.590785	1	False	True	True	False	True
4	0.350833	-0.435486	-1.530560	6.517986e-01	1.264666	-4.682176	-1.577879	1	False	True	False	True	True

Machine Learning Analysis: Classification models

Supervised Machine Learning: Classification

Heart Failure Analysis Prediction

Logistic Regression

- Defining independent and dependent attributes in training and test sets
- Setting up a standard scaler for the features and analyzing it thereafter

```
categorical_val.remove('DEATH_EVENT')
dataset = pd.get_dummies(df, columns = categorical_val)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
col_to_scale = ['age', 'creatinine_phosphokinase', 'ejection_fraction', 'platelets', 'serum_creatinine', 'serum_sodium', 'time']
dataset[col_to_scale] = sc.fit_transform(dataset[col_to_scale])
```

Logistic Regression

	count	mean	std	min	25%	50%	75%	max
age	299.0	5.703353e-16	1.001676	-1.754448	-0.828124	-0.070223	0.771889	2.877170
creatinine_phosphokinase	299.0	0.000000e+00	1.001676	-0.576918	-0.480393	-0.342574	0.000166	7.514640
ejection_fraction	299.0	-3.267546e-17	1.001676	-2.038387	-0.684180	-0.007077	0.585389	3.547716
platelets	299.0	7.723291e-17	1.001676	-2.440155	-0.520870	-0.013908	0.411120	6.008180
serum_creatinine	299.0	1.425838e-16	1.001676	-0.865509	-0.478205	-0.284552	0.005926	7.752020
serum_sodium	299.0	-8.673849e-16	1.001676	-5.363206	-0.595996	0.085034	0.766064	2.582144
time	299.0	-1.901118e-16	1.001676	-1.629502	-0.739000	-0.196954	0.938759	1.997038
DEATH_EVENT	299.0	3.210702e-01	0.467670	0.000000	0.000000	0.000000	1.000000	1.000000

Logistic Regression

Splitting variables into training and test sets

```
X = dataset.drop('DEATH_EVENT', axis=1)
y = dataset.DEATH_EVENT
```

✓ 0.0s

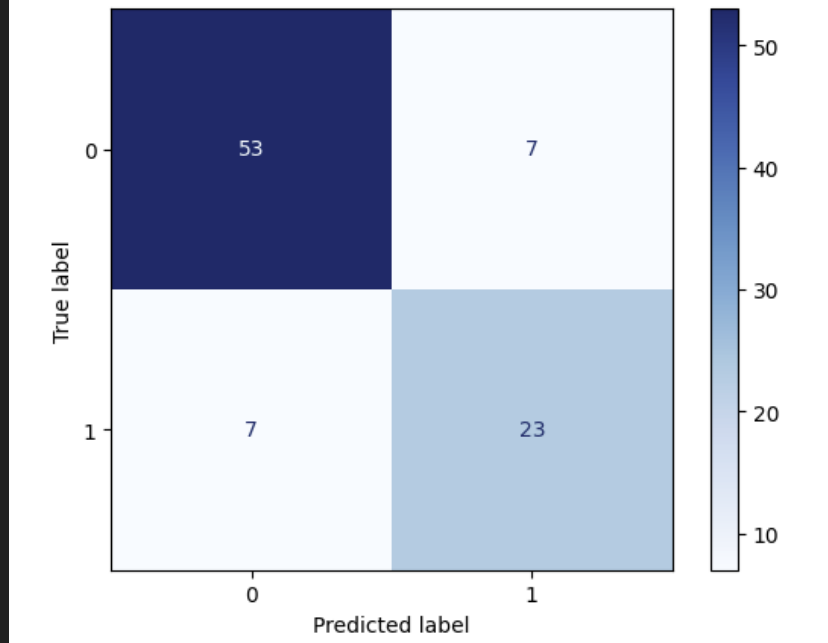
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=25)
```

✓ 0.0s

Logistic Regression

```python
from sklearn.linear_model import LogisticRegression

# Standard logistic regression
lr = LogisticRegression()
lr.fit(X_train, y_train)

# Predicting on the test set
y_pred_0 = lr.predict(X_test)

# Generating the classification report as a DataFrame
clf_report = pd.DataFrame(classification_report(y_test, y_pred_0, output_dict=True))

# Display the classification report
print(clf_report)

# Confusion matrix for KNN predictions
cm = confusion_matrix(y_test, y_pred_0)

# Create the confusion matrix display
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=lr.classes_)

# Plot the confusion matrix
disp.plot(cmap='Blues')  # You can adjust the color map to your liking
plt.grid(False)  # Remove grid
plt.show()  # Display the plot
```

	0	1	accuracy	macro avg	weighted avg
precision	0.883333	0.766667	0.844444	0.825	0.844444
recall	0.883333	0.766667	0.844444	0.825	0.844444
f1-score	0.883333	0.766667	0.844444	0.825	0.844444
support	60.000000	30.000000	0.844444	90.000	90.000000



KNN Classifier

Supervised Machine Learning: Classification

Heart Failure Analysis Prediction

KNN Classifier

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report, f1_score, ConfusionMatrixDisplay

max_k = 40
f1_scores = list()
error_rates = list() # 1-accuracy

for k in range(1, max_k):

    knn = KNeighborsClassifier(n_neighbors=k, weights='distance')
    knn = knn.fit(X_train, y_train)

    y_pred = knn.predict(X_test)
    f1 = f1_score(y_pred, y_test)
    f1_scores.append((k, round(f1_score(y_test, y_pred), 4)))
    error = 1-round(accuracy_score(y_test, y_pred), 4)
    error_rates.append((k, error))

f1_results = pd.DataFrame(f1_scores, columns=['K', 'F1 Score'])
error_results = pd.DataFrame(error_rates, columns=['K', 'Error Rate'])

# Get minimum error id
min_error_id = error_results['Error Rate'].idxmin()

# Get Best K
error_results.loc[min_error_id]
```

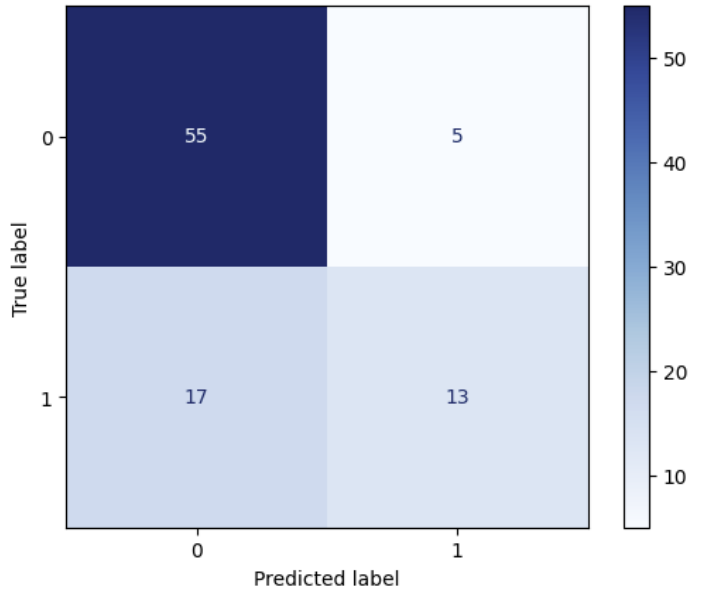
```
K             6.0000
Error Rate    0.2444
Name: 5, dtype: float64
```

KNN Classifier

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, accuracy_score
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Initialize the KNN classifier with n_neighbors=6 and 'distance' weights
knn = KNeighborsClassifier(n_neighbors=6, weights='distance')

# Fit the model to the training data
knn.fit(X_train, y_train)

# Make predictions on the test set
y_pred = knn.predict(X_test)

# Generate the classification report as a DataFrame
KNN_report = pd.DataFrame(classification_report(y_test, y_pred, output_dict=True))

# Print classification report
print("Decision Tree Classifier Classification Report:")
print(KNN_report)

# Confusion matrix for KNN predictions
cm = confusion_matrix(y_test, y_pred, labels=knn.classes_)

# Create the confusion matrix display
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=knn.classes_)

# Plot the confusion matrix
disp.plot(cmap='Blues')  # You can adjust the color map to your liking
plt.grid(False)  # Remove grid
plt.show()  # Display the plot
```

Decision Tree Classifier Classification Report:

	0	1	accuracy	macro avg	weighted avg
precision	0.763889	0.722222	0.755556	0.743056	0.750000
recall	0.916667	0.433333	0.755556	0.675000	0.755556
f1-score	0.833333	0.541667	0.755556	0.687500	0.736111
support	60.000000	30.000000	0.755556	90.000000	90.000000



SVM Classifier

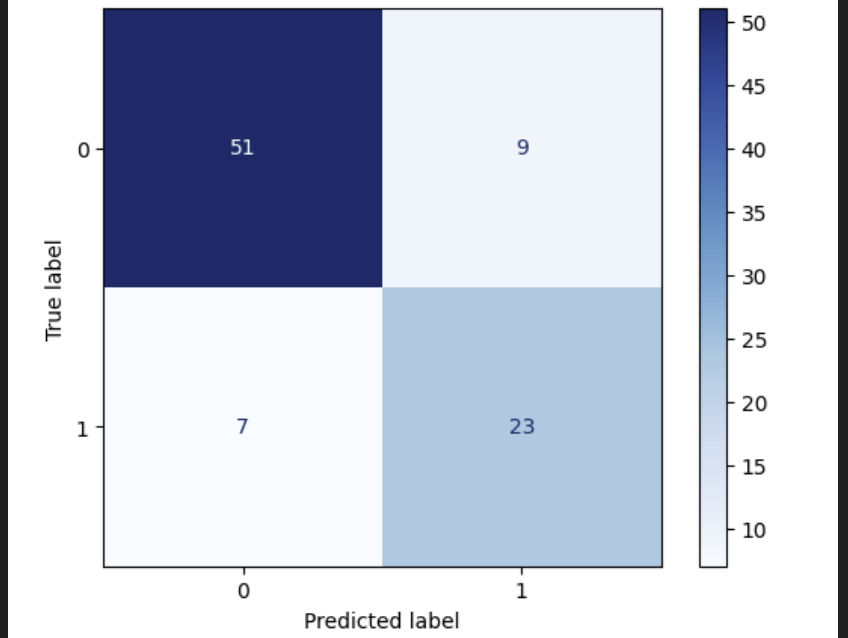
Supervised Machine Learning: Classification

Heart Failure Analysis Prediction

SVM Classifier

```python
from sklearn.svm import SVC
from sklearn.metrics import classification_report

svc = SVC()

# Fit the model on the training data
SVC_cl = svc.fit(X_train, y_train)

# Make predictions on the test set
y_pred = SVC_cl.predict(X_test)

# Generate the classification report as a DataFrame
SVC_cl_report = pd.DataFrame(classification_report(y_test, y_pred, output_dict=True))

# Display the classification report
print(SVC_cl_report)

# Confusion matrix for SVC predictions
cm = confusion_matrix(y_test, y_pred, labels=SVC_cl.classes_)

# Create the confusion matrix display
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=SVC_cl.classes_)

# Plot the confusion matrix
disp.plot(cmap='Blues')  # You can adjust the color map to your liking
plt.grid(False)  # Remove grid
plt.show()  # Display the plot
```

	0	1	accuracy	macro avg	weighted avg
precision	0.879310	0.718750	0.822222	0.799030	0.825790
recall	0.850000	0.766667	0.822222	0.808333	0.822222
f1-score	0.864407	0.741935	0.822222	0.803171	0.823583
support	60.000000	30.000000	0.822222	90.000000	90.000000

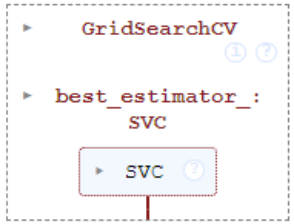


SVM Classifier: Grid Search

```python
#Hyperparameter tuning with GridSearchCV
params_grid = {
    'C': [1, 10, 100],
    'kernel': ['poly', 'rbf', 'sigmoid']
}
grid_search = GridSearchCV(SVC(), param_grid=params_grid, scoring='f1', cv=5, verbose=1)
grid_search.fit(X_train, y_train)
```

✓ 0.3s

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
```




```python
# Get the best model from GridSearchCV
best_model = grid_search.best_estimator_
print("Best parameters:", grid_search.best_params_)
```

✓ 0.0s

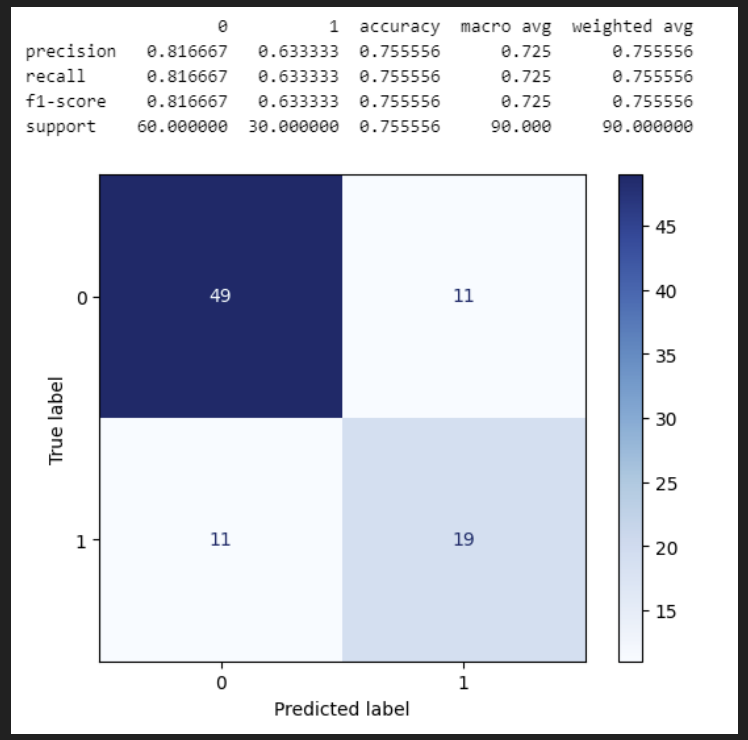
```
Best parameters: {'C': 1, 'kernel': 'sigmoid'}
```

SVM Classifier: Grid Search

```
# Generate and display the classification report
SVC_cl_report = pd.DataFrame(classification_report(y_test, preds, output_dict=True))
print(SVC_cl_report)

# Confusion matrix for best model
cm = confusion_matrix(y_test, preds, labels=best_model.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=best_model.classes_)
disp.plot(cmap='Blues')
plt.grid(False)
plt.show()
```

✓ 0.1s

	0	1	accuracy	macro avg	weighted avg
precision	0.816667	0.633333	0.755556	0.725	0.755556
recall	0.816667	0.633333	0.755556	0.725	0.755556
f1-score	0.816667	0.633333	0.755556	0.725	0.755556
support	60.000000	30.000000	0.755556	90.000	90.000000



Decision Tree Classifier

Supervised Machine Learning: Classification

Heart Failure Analysis Prediction

Decision Tree Classifier

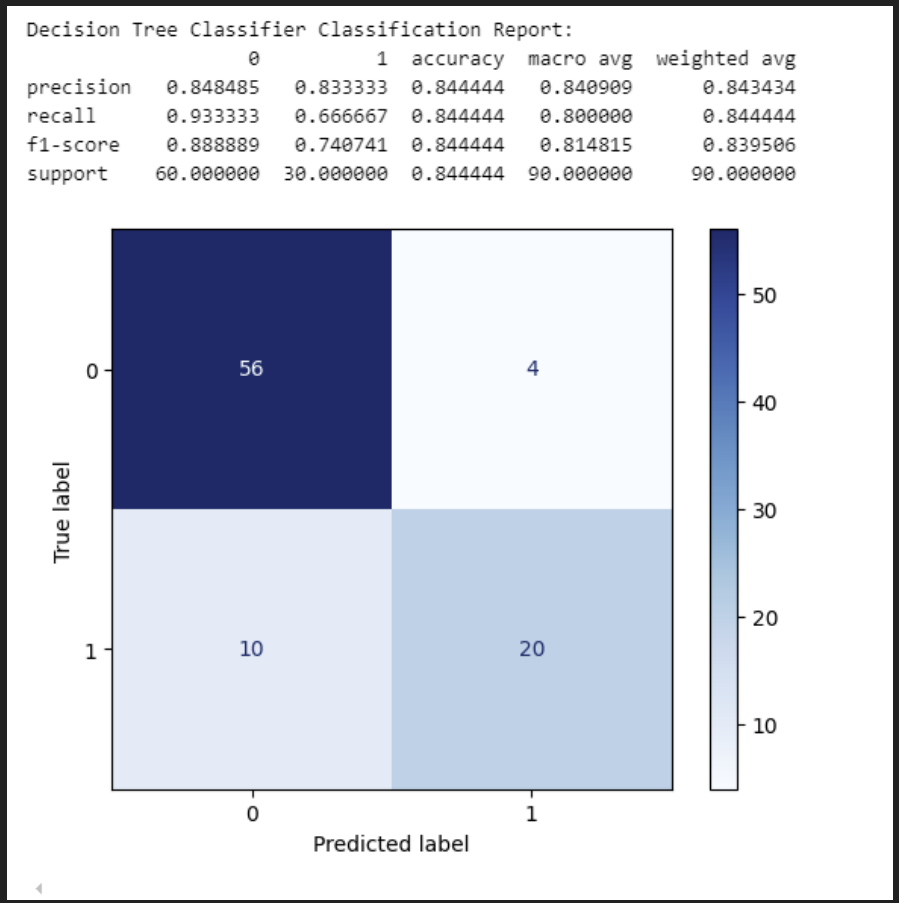
```
from sklearn.tree import DecisionTreeClassifier

# Initialize the Decision Tree Classifier (you can adjust max_depth and other parameters)
dt_clf = DecisionTreeClassifier(max_leaf_nodes=3, random_state=42, criterion='entropy')

# Fit the model on the training data
dt_clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred_dt = dt_clf.predict(X_test)

# Generate the classification report as a DataFrame
dt_clf_report = pd.DataFrame(classification_report(y_test, y_pred_dt, output_dict=True))

# Display the classification report
print("Decision Tree Classifier Classification Report:")
print(dt_clf_report)

# Generate and plot the confusion matrix
cm_dt = confusion_matrix(y_test, y_pred_dt, labels=dt_clf.classes_)

# Create the confusion matrix display
disp_dt = ConfusionMatrixDisplay(confusion_matrix=cm_dt, display_labels=dt_clf.classes_)

# Plot the confusion matrix
disp_dt.plot(cmap='Blues')  # You can adjust the color map to your liking
plt.grid(False)  # Remove grid
plt.show()  # Display the plot
```

✓ 0.3s

Decision Tree Classifier

Decision Tree Classifier Classification Report:

	0	1	accuracy	macro avg	weighted avg
precision	0.848485	0.833333	0.844444	0.840909	0.843434
recall	0.933333	0.666667	0.844444	0.800000	0.844444
f1-score	0.888889	0.740741	0.844444	0.814815	0.839506
support	60.000000	30.000000	0.844444	90.000000	90.000000



Decision Tree Classifier: Grid Search

```
params_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [5, 10, 15, 20],
    'min_samples_leaf': [1, 2, 5]
}

model = DecisionTreeClassifier(random_state=42)
grid_search = GridSearchCV(estimator = model,
                           param_grid = params_grid,
                           scoring='f1',
                           cv = 5, verbose = 1)
grid_search.fit(X_train, y_train.values.ravel())
best_params = grid_search.best_params_
best_params
```

✓ 0.6s Open 'best_params' in Data Wrangler

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits

{'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 1}
```

Decision Tree Classifier: Grid Search

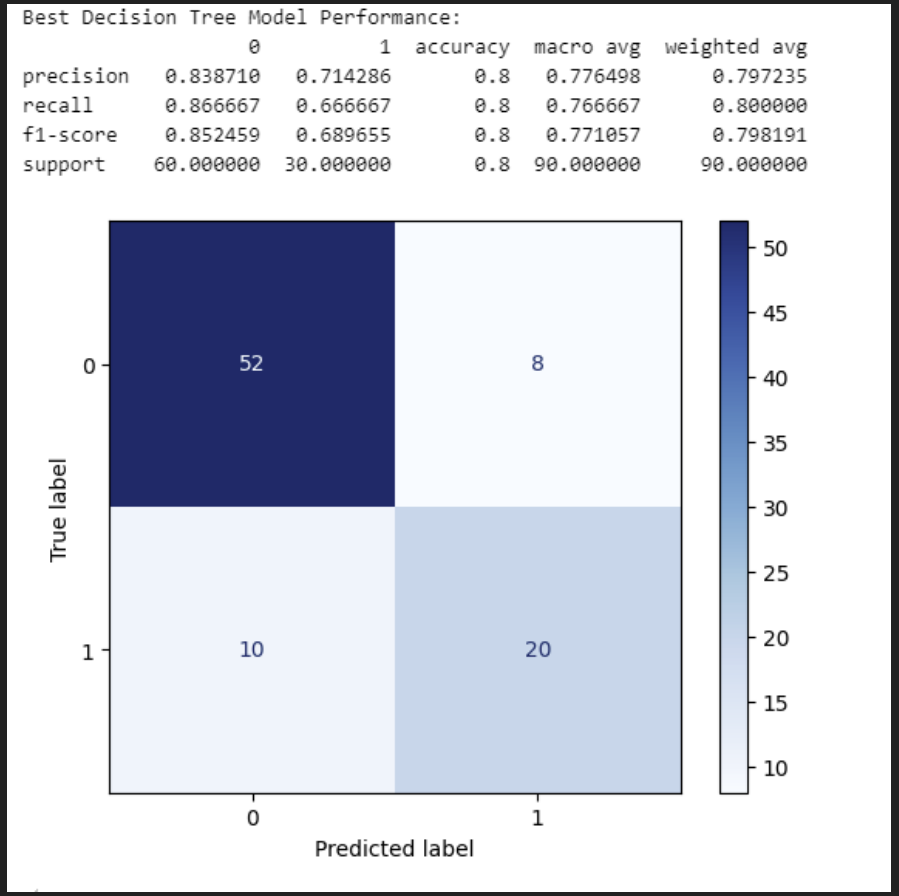
```
# Retrieve the best model from GridSearchCV
best_dt_clf = grid_search.best_estimator_

# Make predictions on the test set
y_pred = best_dt_clf.predict(X_test)

# Evaluate model performance
print("Best Decision Tree Model Performance:")
dt_report = pd.DataFrame(classification_report(y_test, y_pred, output_dict=True))
print(dt_report)

# Confusion matrix
cm = confusion_matrix(y_test, y_pred, labels=best_dt_clf.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=best_dt_clf.classes_)
disp.plot(cmap='Blues')
plt.grid(False)
plt.show()
```

✓ 0.1s

Decision Tree Classifier: Grid Seach

Best Decision Tree Model Performance:

	0	1	accuracy	macro avg	weighted avg
precision	0.838710	0.714286	0.8	0.776498	0.797235
recall	0.866667	0.666667	0.8	0.766667	0.800000
f1-score	0.852459	0.689655	0.8	0.771057	0.798191
support	60.000000	30.000000	0.8	90.000000	90.000000

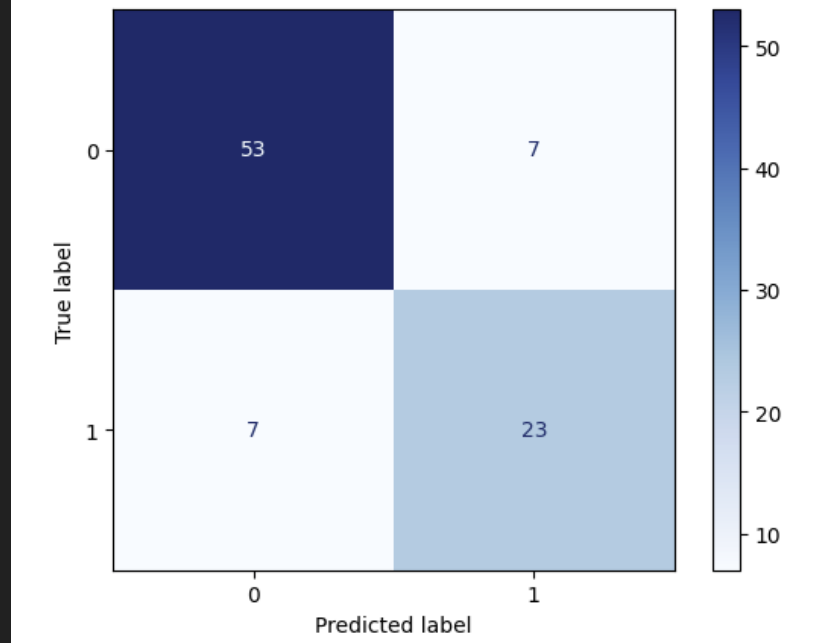


Model Comparison

While both models have similar accuracy, the logistic regression model demonstrates a better balance in performance across both classes and a lower number of false negatives, making it slightly more suitable for this specific scenario :

1. Logistic Model
2. Decision Tree
3. SVM
4. KNN

	0	1	accuracy	macro avg	weighted avg
precision	0.883333	0.766667	0.844444	0.825	0.844444
recall	0.883333	0.766667	0.844444	0.825	0.844444
f1-score	0.883333	0.766667	0.844444	0.825	0.844444
support	60.000000	30.000000	0.844444	90.000	90.000000



Conclusion

Supervised Machine Learning: Classification

Heart Failure Analysis Prediction

Conclusion

Interpretability: Logistic regression is often easier to interpret, as the coefficients associated with the features provide insights into their importance. Decision trees can also be interpreted but might become complex with many levels.

Model Complexity: Decision trees can be more prone to overfitting, especially if not pruned properly.1 Logistic regression is generally simpler and less likely to overfit.

Dataset Characteristics: The performance of these models can vary depending on the specific dataset and its characteristics (e.g., size, feature types, class balance).

Thank you

IBM Machine Learning Professional Certificate

Supervised Machine Learning: Classification

Shaula Marquez