Assignment-2

April 30, 2024

0.1 Instructions:

In this assignment, we will explore the fundamentals of classification tasks, aiming to leverage various tools and techniques learned throughout the course. Our focus will be on understanding the data, identifying meaningful patterns, and employing appropriate classification algorithms to predict the target variable accurately. Through this report, we aim to present our findings, insights, and potential next steps in the classification process.

0.2 Import the required libraries

[61]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

0.3 Importing the Dataset

[62]:
```
df = pd.read_csv("data/heart_failure_clinical_records_dataset.csv")
```

1 1. About the Data

This dataset contains the medical records of 299 patients who had heart failure, collected during their follow-up period, where each patient profile has 13 clinical features.

[63]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 299 entries, 0 to 298
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   age                       299 non-null    float64
 1   anaemia                   299 non-null    int64
 2   creatinine_phosphokinase  299 non-null    int64
 3   diabetes                  299 non-null    int64
 4   ejection_fraction         299 non-null    int64
 5   high_blood_pressure       299 non-null    int64
```

```
 6   platelets                 299 non-null    float64
 7   serum_creatinine          299 non-null    float64
 8   serum_sodium              299 non-null    int64
 9   sex                       299 non-null    int64
 10  smoking                   299 non-null    int64
 11  time                      299 non-null    int64
 12  DEATH_EVENT               299 non-null    int64
dtypes: float64(3), int64(10)
memory usage: 30.5 KB
```

[64]:
```
df.head(5)
```

[64]:
```
    age  anaemia  creatinine_phosphokinase  diabetes  ejection_fraction  \
0  75.0        0                       582         0                 20
1  55.0        0                      7861         0                 38
2  65.0        0                       146         0                 20
3  50.0        1                       111         0                 20
4  65.0        1                       160         1                 20

   high_blood_pressure  platelets  serum_creatinine  serum_sodium  sex  \
0                    1  265000.00               1.9           130    1
1                    0  263358.03               1.1           136    1
2                    0  162000.00               1.3           129    1
3                    0  210000.00               1.9           137    1
4                    0  327000.00               2.7           116    0

   smoking  time  DEATH_EVENT
0        0     4            1
1        0     6            1
2        1     7            1
3        0     7            1
4        0     8            1
```

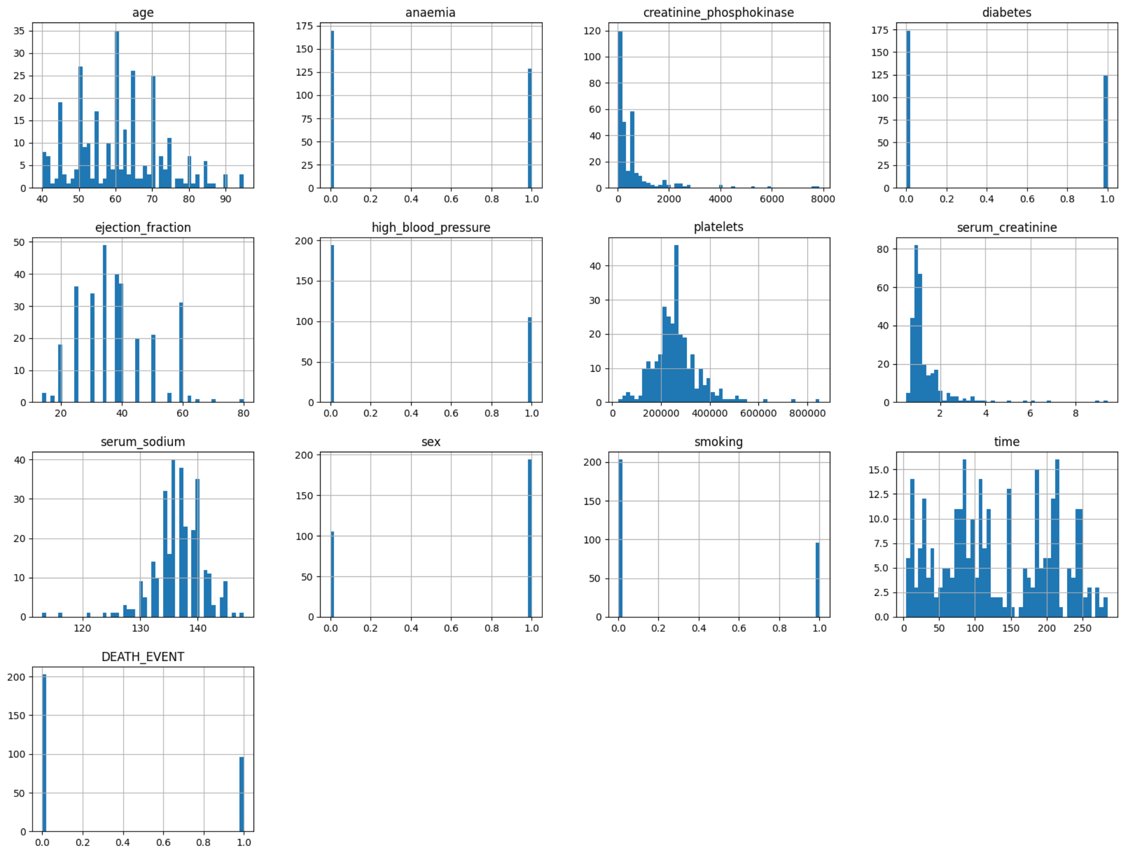
[65]:
```
df.describe()
```

[65]:
```
              age     anaemia  creatinine_phosphokinase    diabetes  \
count  299.000000  299.000000                299.000000  299.000000
mean    60.833893    0.431438                581.839465    0.418060
std     11.894809    0.496107                970.287881    0.494067
min     40.000000    0.000000                 23.000000    0.000000
25%     51.000000    0.000000                116.500000    0.000000
50%     60.000000    0.000000                250.000000    0.000000
75%     70.000000    1.000000                582.000000    1.000000
max     95.000000    1.000000               7861.000000    1.000000

       ejection_fraction  high_blood_pressure      platelets  \
count         299.000000           299.000000     299.000000
mean           38.083612             0.351171  263358.029264
```

```
std          11.834841          0.478136   97804.236869
min          14.000000          0.000000   25100.000000
25%          30.000000          0.000000  212500.000000
50%          38.000000          0.000000  262000.000000
75%          45.000000          1.000000  303500.000000
max          80.000000          1.000000  850000.000000

       serum_creatinine  serum_sodium         sex    smoking        time  \
count         299.00000    299.000000  299.000000  299.00000  299.000000
mean            1.39388    136.625418    0.648829    0.32107  130.260870
std             1.03451      4.412477    0.478136    0.46767   77.614208
min             0.50000    113.000000    0.000000    0.00000    4.000000
25%             0.90000    134.000000    0.000000    0.00000   73.000000
50%             1.10000    137.000000    1.000000    0.00000  115.000000
75%             1.40000    140.000000    1.000000    1.00000  203.000000
max             9.40000    148.000000    1.000000    1.00000  285.000000

       DEATH_EVENT
count    299.00000
mean       0.32107
std        0.46767
min        0.00000
25%        0.00000
50%        0.00000
75%        1.00000
max        1.00000
```

1.0.1 Draw the plots for visualization.

[66]:
```
df.hist(bins=50, figsize=(20,15))
plt.show()
```



1.0.2 Find the missing value.

[67]:
```
df.columns[df.isnull().any()]
df.isnull().sum()
```

[67]:
```
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
```

Drop the outlier values.

[68]:
```python
def remove_outliers(df):
    # Create a new DataFrame to hold the filtered data
    filtered_df = df.copy()

    # Iterate over each column in the DataFrame
    for column in df.select_dtypes(include=['number']).columns:
        Q1 = df[column].quantile(0.25)
        Q3 = df[column].quantile(0.75)
        IQR = Q3 - Q1

        # Define the bounds for outliers
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Filter the DataFrame without outliers
        filtered_df = filtered_df[(filtered_df[column] >= lower_bound) & 
 ↪(filtered_df[column] <= upper_bound)]

    return filtered_df
```

[69]:
```python
df = remove_outliers(df)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 224 entries, 0 to 298
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   age                       224 non-null    float64
 1   anaemia                   224 non-null    int64
 2   creatinine_phosphokinase  224 non-null    int64
 3   diabetes                  224 non-null    int64
 4   ejection_fraction         224 non-null    int64
 5   high_blood_pressure       224 non-null    int64
 6   platelets                 224 non-null    float64
 7   serum_creatinine          224 non-null    float64
 8   serum_sodium              224 non-null    int64
 9   sex                       224 non-null    int64
 10  smoking                   224 non-null    int64
 11  time                      224 non-null    int64
 12  DEATH_EVENT               224 non-null    int64
dtypes: float64(3), int64(10)
memory usage: 24.5 KB
```

2 2. Objectives

The primary purpose of this type of dataset is to perform various analyses using machine learning techniques. These analyses can include:

1. **Classification:** Predicting whether patients will survive or not. For example, identifying which patients are at high risk can help in developing prioritized intervention strategies for those patients.
2. **Regression:** Predicting how long patients can survive, which is useful for estimating a continuous output value. This can be beneficial in planning treatment processes and in the more effective distribution of resources.

3 3. Classification Regression Models

3.0.1 Train-Test Split

[70]:
```
X = df.drop("DEATH_EVENT", axis=1)
y = df["DEATH_EVENT"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

3.0.2 K-Nearest Neighbor (KNN) Model

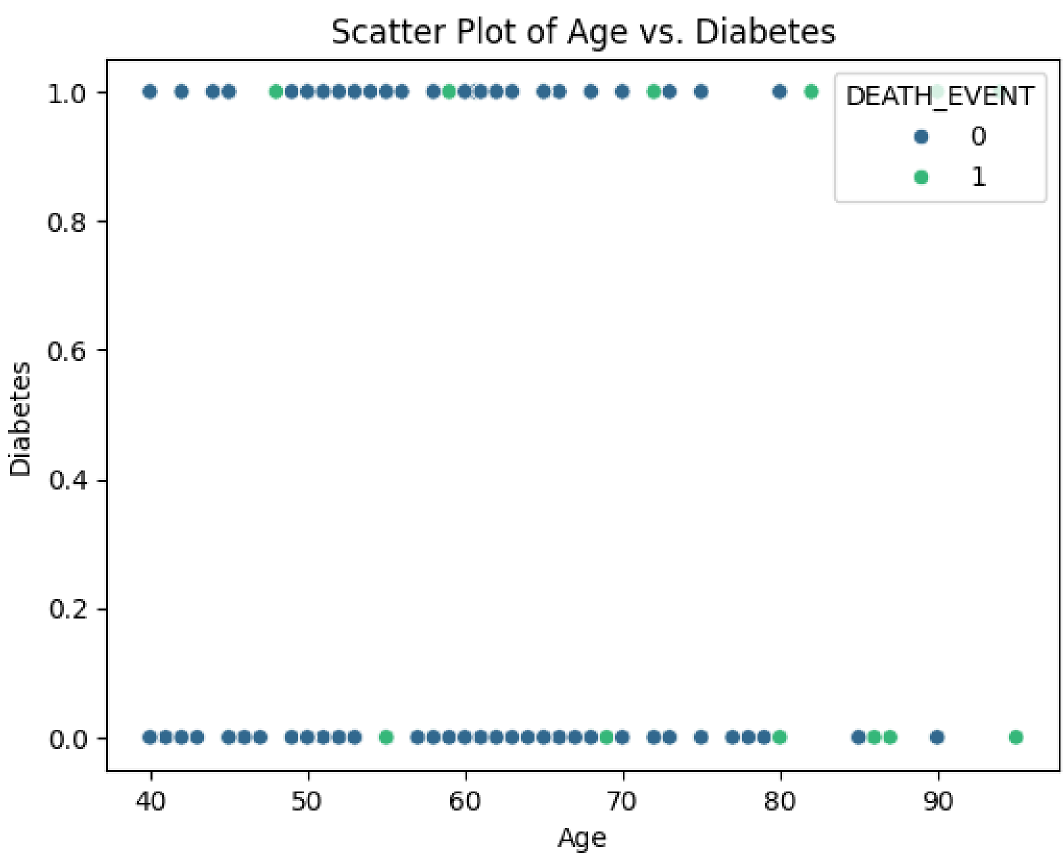
[71]:
```
# Scaling the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Creating and training the KNN model
knn = KNeighborsClassifier(n_neighbors=5)  # K = 5 chosen, you can choose any␣
↪number you want
knn.fit(X_train_scaled, y_train)

# Making predictions on the test set
y_pred = knn.predict(X_test_scaled)

# Calculating accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"K-NN Accuracy: {accuracy:.2f}")

# Generating classification report
report = classification_report(y_test, y_pred)
print("K-NN Classification Report:")
print(report)
```

```
K-NN Accuracy: 0.80
K-NN Classification Report:
              precision    recall  f1-score   support
```

```
              0       0.79      1.00      0.88        34
              1       1.00      0.18      0.31        11

       accuracy                           0.80        45
      macro avg       0.90      0.59      0.60        45
   weighted avg       0.84      0.80      0.74        45
```

[72]:

```python
sns.scatterplot(data=df, x='age', y='diabetes', hue='DEATH_EVENT',␣
 ↪palette='viridis')
plt.title('Scatter Plot of Age vs. Diabetes')
plt.xlabel('Age')
plt.ylabel('Diabetes')
plt.legend(title='DEATH_EVENT', loc='upper right')
plt.show()
```



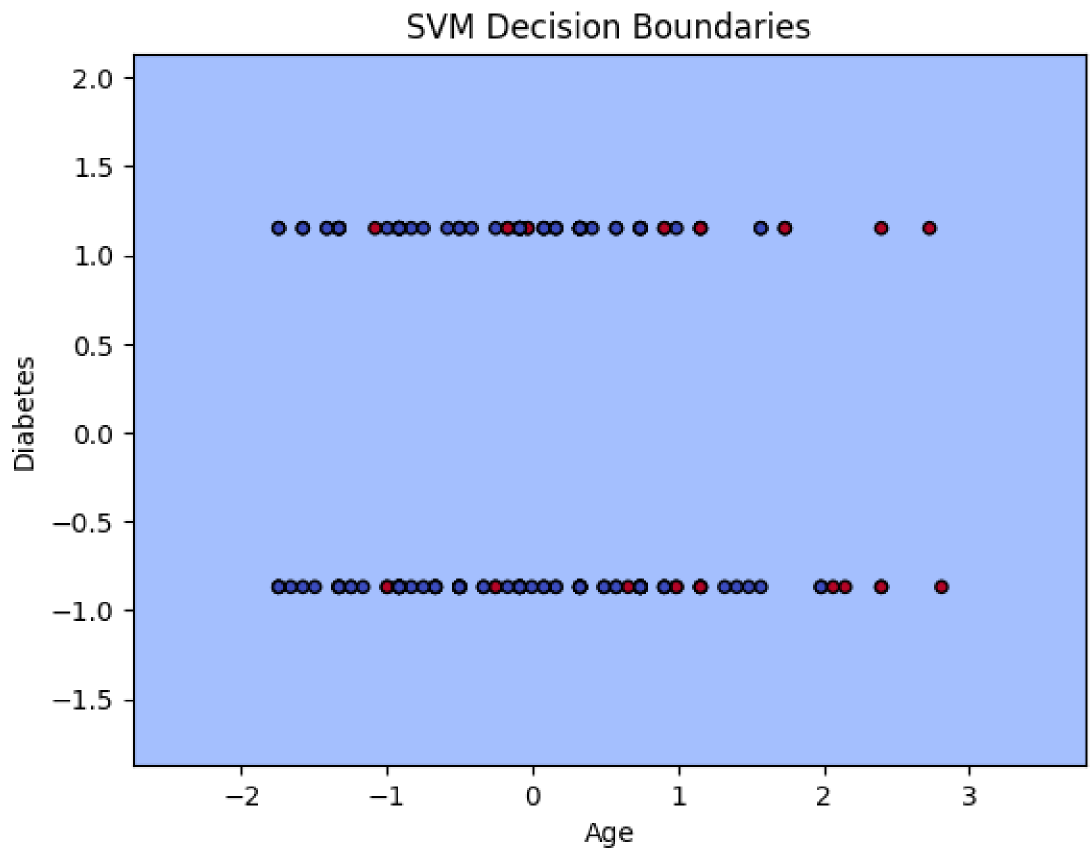
3.0.3 Support Vector Machine Model

[73]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

svm = SVC(kernel='linear')
svm.fit(X_train_scaled, y_train)

y_pred_svm = svm.predict(X_test_scaled)

accuracy_svm = accuracy_score(y_test, y_pred_svm)
print(f"SVM Accuracy: {accuracy_svm:.2f}")

report_svm = classification_report(y_test, y_pred_svm)
print("SVM Classification Report:")
print(report_svm)
```

```
SVM Accuracy: 0.80
SVM Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.94      0.88        34
           1       0.67      0.36      0.47        11

    accuracy                           0.80        45
   macro avg       0.74      0.65      0.67        45
weighted avg       0.78      0.80      0.78        45
```

[74]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_train.iloc[:, [0, 3]])

svm = SVC(kernel='linear')
svm.fit(X_scaled, y_train)

h = .02
x_min, x_max = X_scaled[:, 0].min() - 1, X_scaled[:, 0].max() + 1
y_min, y_max = X_scaled[:, 1].min() - 1, X_scaled[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))
Z = svm.predict(np.c_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)
```

```
plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=y_train, cmap=plt.cm.coolwarm,␣
 ↪s=20, edgecolors='k')
plt.xlabel('Age')
plt.ylabel('Diabetes')
plt.title('SVM Decision Boundaries')
plt.show()
```



3.0.4 Logistic Regression Model

[75]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

X_train, X_test, y_train, y_test = train_test_split(df.
 ↪drop(columns=["DEATH_EVENT"]), df["DEATH_EVENT"], test_size=0.2,␣
 ↪random_state=42)

class LogisticRegressionClassifier:
```

```python
    def __init__(self):
        self.model = LogisticRegression()

    def train(self, X_train, y_train):
        self.model.fit(X_train, y_train)

    def predict(self, X_test):
        return self.model.predict(X_test)

    def evaluate(self, X_test, y_test):
        y_pred = self.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)
        report = classification_report(y_test, y_pred)
        return accuracy, report

classifier = LogisticRegressionClassifier()
classifier.train(X_train, y_train)
accuracy, report = classifier.evaluate(X_test, y_test)
print(f"Logistic Regression Accuracy: {accuracy:.2f}")
print("Logistic Regression Classification Report:")
print(report)
```

```
Logistic Regression Accuracy: 0.87
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.97      0.92        34
           1       0.86      0.55      0.67        11

    accuracy                           0.87        45
   macro avg       0.86      0.76      0.79        45
weighted avg       0.87      0.87      0.86        45
```

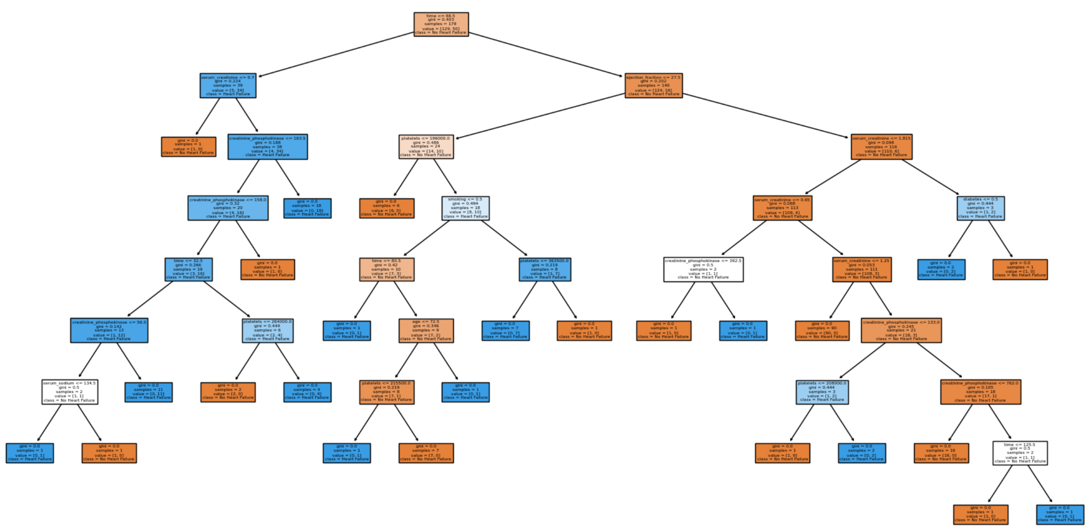
3.0.5 Decision Tree Model

[76]:
```python
decision_tree = DecisionTreeClassifier(random_state=42)
decision_tree.fit(X_train, y_train)

y_pred = decision_tree.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Decision Tree Accuracy: {accuracy:.2f}")

report = classification_report(y_test, y_pred)
print("Decision Tree Classification Report:")
print(report)
```

```
Decision Tree Accuracy: 0.71
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.79      0.81        34
           1       0.42      0.45      0.43        11

    accuracy                           0.71        45
   macro avg       0.62      0.62      0.62        45
weighted avg       0.72      0.71      0.72        45
```

3.0.6 Draw decision tree

[77]:
```python
from sklearn.tree import plot_tree

plt.figure(figsize=(20,10))
plot_tree(decision_tree, filled=True, feature_names=X.columns, class_names=["No Heart Failure", "Heart Failure"])
plt.show()
```



4 4. Insights and key findings

Throughout our analysis, we tested several regression models to identify the best predictor of heart failure. he Logistic Regression model, K-Nearest Neighbor (KNN) Model and Support Vector Machine Model showed high accuracy, with the Logistic Regression model performing slightly better, achieving an accuracy of 87%. Based on the dataset and its attributes, key findings suggest that variables such as age and serum creatinine levels are significant predictors of heart failure.

The dataset reveals that older age and elevated serum creatinine levels strongly indicate the risk of heart failure.

5 5. Next Steps

Future work will focus on implementing cross-validation methods to improve the robustness of our models and experimenting with more sophisticated algorithms, such as Random Forest and Gradient Boosting Machines. Moreover, by integrating more detailed information, such as patients' medical histories and lifestyle details, we can enhance the predictive accuracy of our models.