FINAL REPORT

CS361L-ARTIFICIAL INTELLIGENCE

SUBMITTED TO:-

SIR QAISER ABBAS

DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF ENGINEERING & TECHNOLOGY, LAHORE

Contents

Linear Regression:

- ## Data description:

Ten baseline variables, age, sex, body mass index, average blood pressure, and six blood serum measurements were obtained for each of n = 442 diabetes patients, as well as the response of interest, a quantitative measure of disease progression one year after baseline.

The dataset contains total of 442 instances. There are ten attributes including age, sex, bmi, bp, s1, s2, s3, s4, s5, and s6. Age is age in years, sex is the gender, bmi is body mass index, bp is average blood pressure, s1 is total serum cholesterol, s2 is low-density lipoproteins, s3 is high-density lipoproteins, s4 is total cholesterol per HDL, s5 is serum triglycerides level, and s6 is blood sugar level.

The variables are mean centered and scaled by standard deviation multiplied by the n samples (442), which means that the sum of square of each column is equal to 1.

The dataset is imported from scikit-learn at:

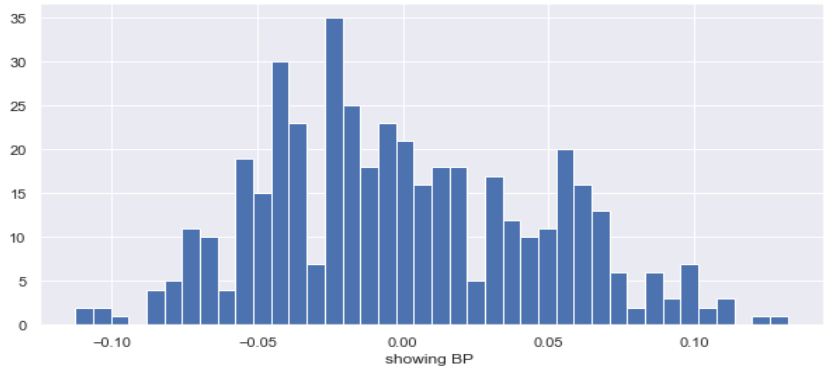
https://scikit-learn.org/stable/datasets/toy_dataset.html

- ## Data Preprocessing:

The dataset taken from the repository is complete and preprocessed already. No further preprocessing was done on dataset as it was not found in the raw form. Dataset was cleaned, integrated, reduced, and transformed by default. So, the dataset was in usable form and valid to be used for Machine Learning Algorithms.

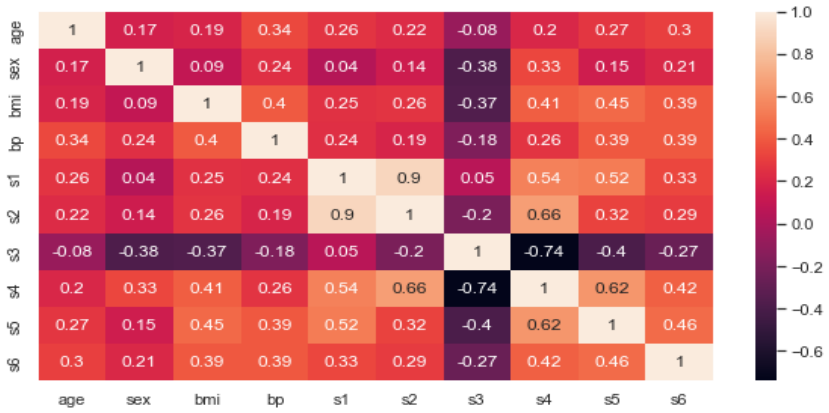
- ## Data Visualization:

The graph down below shows the attribute of "bp"



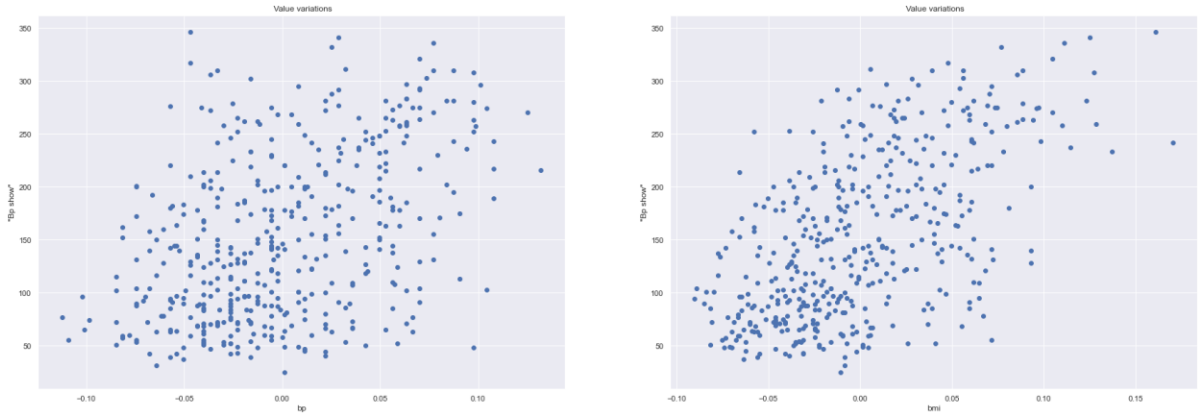
Heat Map:

Below is the heat map for the data, and feature names of the data set. The heat map shows the correlation between the attributes.



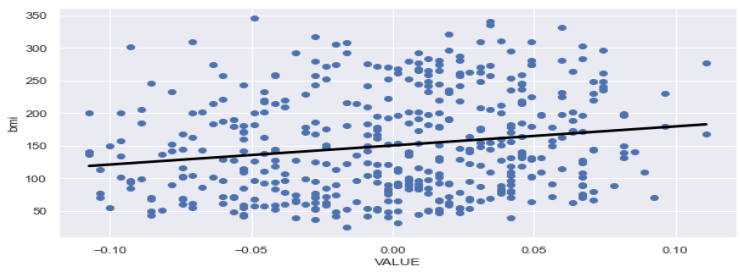
Scatter Plot:

The first scatter plot is about bp. The y-axis contains the bp values, while the x-axis shows the value variation. The second plot contains the points for attribute “bmi”.



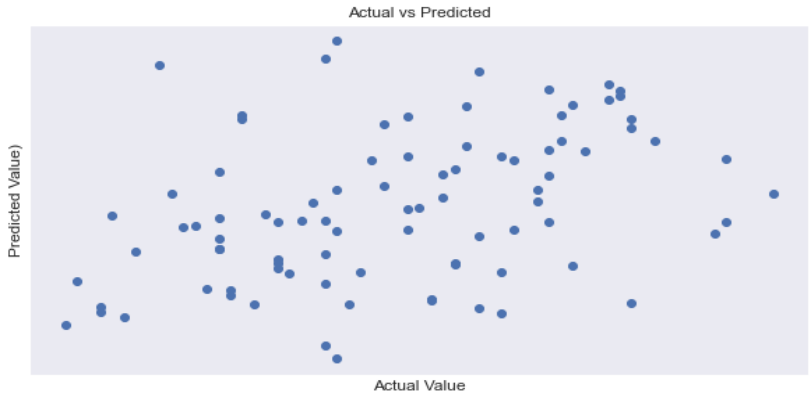
Regression Plot:

The graph down below is the plot between bmi on y-axis and VALUE on x-axis.



Actual Vs. Predicted Values:

The last plot is about the actual values and the predicted values.



- **Model Implementation Details:**

Introduction:

Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data. ... A linear regression line has an equation of the form **Y = a + bX**, where X is the explanatory variable and Y is the dependent variable.

Model implementation:

1. In the first line, the libraries required for the implementation of the model are imported such as numpy, pandas, etc.
2. Also, the required models are also imported such as datasets, LinearRegression, train_test_split, and mean_Squared_error.
3. Next, the dataset is loaded. And the data shape, which is the matrix of rows and columns of dataset, the keys which are attributes of the dictionary, and the feature names are shown as output.
4. The description of dataset is also shown as output through the function dataset.DESCR
5. In the next line, the dataFrame which is the SQL table or 2-D structure which shows the data is printed. The target value is also specified here termed as "VALUE". It is the measure of disease progression one year after the baseline. The "VALUE" is the 11 column in the dataset.
6. Next, the function ".isnull().sum()" is used to show the null values (if any) present in the dataset. The values are also described in the table in terms of mean, min, max, std, count for each attribute.
7. The bar graph is then shown for (any) single independent attribute. And also, a heat map is constructed by the function of heatmap. The heat map shows the

correlation matrix among different attributes. Next, scatter plots are also generated using scatter function.

8. After the scatter plot, the rows and columns for x being the independent attributes and y being the target value. The numpy array is constructed and reshaped.
9. Then, the train_test_finction is applied on the dataset to divide the data for testing and training phase.
10. Finally, the linear regression model is applied on the x and y train data. Prediction is made on y train. RMSE function is applied to get the average value of the errors.
11. Next, the model is applied for the testing data and the same steps are repeated.
12. The plot is made for the linear regression prediction between the bmi and the target value. And lastly, the scatter plot is generated for the actual vs the predicted values.

- **Results:**

The linear regression model is applied on the testing and training data. The model performance for training dataset is

Root mean squared error of 0.04192 and r^2 score of 0.18

Similarly, the model is applied on the testing data where the performance is as follows;

Root mean squared error of 0.048125 and r^2 score of 0.14

- **Code:**

```
import numpy as np

import pandas as pd

#Visualization Libraries

import seaborn as sns

import matplotlib.pyplot as plt


#To plot the graph embedded in the notebook

%matplotlib inline

rom sklearn import datasets

from sklearn.linear_model import LinearRegression

from sklearn.model_selection import train_test_split, cross_val_score

from sklearn.metrics import mean_squared_error

diabetes = datasets.load_diabetes()
```

```
print(type(diabetes))
print('\n')
print(diabetes.keys())
print('\n')
print(diabetes.data.shape)
print('\n')
print(diabetes.feature_names)
print(diabetes.DESCR)
bos = pd.DataFrame(diabetes.data, columns = diabetes.feature_names)
bos['VALUE'] = diabetes.target
print(bos.head())
bos.isnull().sum()
print(bos.describe())
sns.set(rc={'figure.figsize':(10.0,5.0)})
plt.hist(bos['bp'], bins=40)
plt.xlabel("showing BP ")
plt.show()
bos_1 = pd.DataFrame(diabetes.data, columns = diabetes.feature_names)
correlation_matrix = bos_1.corr().round(2)
sns.heatmap(data=correlation_matrix, annot=True)
plt.figure(figsize=(30, 10))

features = ['bp', 'bmi']
target = bos['VALUE']

for i, col in enumerate(features):
    plt.subplot(1, len(features) , i+1)
    x = bos[col]
```

```
    y = target
    plt.scatter(x, y, marker='o')
    plt.title(" Value variations")
    plt.xlabel(col)
    plt.ylabel("'Bp show"')
X_attributes = bos.age
y_target = bos.VALUE
X_attributes = np.array(X_attributes).reshape(-1,1)
y_target = np.array(y_target).reshape(-1,1)
print(X_attributes.shape)
print(y_target.shape)
X_train_1, X_test_1, Y_train_1, Y_test_1 = train_test_split(X_attributes, y_target, test_size =
0.3, random_state=5)
print(X_train_1.shape)
print(X_test_1.shape)
print(Y_train_1.shape)
print(Y_test_1.shape)
reg_1 = LinearRegression()
reg_1.fit(X_train_1, Y_train_1)
y_train_predict_1 = reg_1.predict(X_train_1)
rmse = (np.sqrt(mean_squared_error(Y_train_1, y_train_predict_1)))
r2 = round(reg_1.score(X_train_1, Y_train_1),2)
print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
print("\n")
y_pred_1 = reg_1.predict(X_test_1)
rmse = (np.sqrt(mean_squared_error(Y_test_1, y_pred_1)))
```

```
r2 = round(reg_1.score(X_test_1, Y_test_1),2)
print("The model performance for training set")
print("--------------------------------------")
print("Root Mean Squared Error: {}".format(rmse))
print("R^2: {}".format(r2))
print("\n")
prediction_space = np.linspace(min(X_attributes), max(X_attributes)).reshape(-1,1)
plt.scatter(X_attributes,y_target)
plt.plot(prediction_space, reg_1.predict(prediction_space), color = 'black', linewidth = 3)
plt.ylabel('bmi')
plt.xlabel('VALUE')
plt.show()
X = bos.drop('age', axis = 1)
y = bos['age']
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2, random_state=42)
reg_all = LinearRegression()
reg_all.fit(X_train, y_train)
# model evaluation for training set
y_train_predict = reg_all.predict(X_train)
rmse = (np.sqrt(mean_squared_error(y_train, y_train_predict)))
r2 = round(reg_all.score(X_train, y_train),2)
print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
print("\n")
y_pred = reg_all.predict(X_test)
rmse = (np.sqrt(mean_squared_error(y_test, y_pred)))
```

```
r2 = round(reg_all.score(X_test, y_test),2)
print("The model performance for training set")
print("--------------------------------------")
print("Root Mean Squared Error: {}".format(rmse))
print("R^2: {}".format(r2))
print("\n")
plt.scatter(y_test, y_pred)
plt.xlabel("Actual Value")
plt.ylabel("Predicted Value)")
plt.xticks(range(0, int(max(y_test)),2))
plt.yticks(range(0, int(max(y_test)),2))
plt.title("Actual vs Predicted")
```

Logistic Regression

- ## Dataset Description:

The dataset is about heart attack prediction. The prediction is based on values of 0 and 1 where 0 shows death. The output i-e the DEATH_EVENT is based on the features like HIGH BLOOD PRESSURE, DIABETES, SMOKING, etc.

The output, DEATH_EVENT is the dependent variable depending on the attributes which are independent. The dataset contains real values for each attribute where the observation made falls in the range of {0, 1}.

The dataset contains 13 columns representing attributes of AGE, ANEMIA, CREATANINE, DIABETES, EJECTION_FRACTION, HIGH_BLOOD_PRESSURE, PLATELETS, SERUM_CREATANINE, SERUM_SODIUM, SEX, SMOKING, TIME, DEATH_EVENT. It has a total of 300 rows.

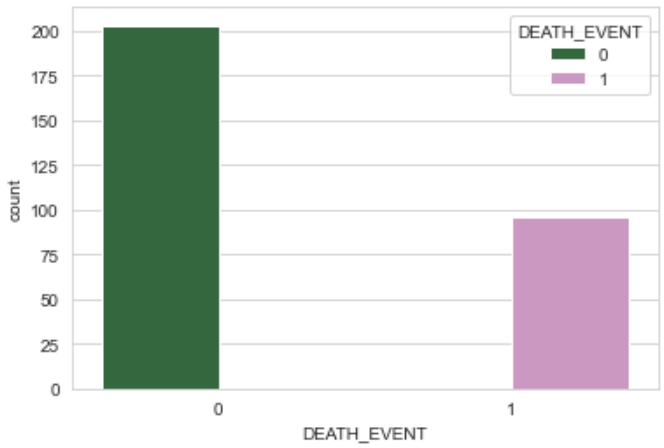
The source of the dataset is UCI Machine Learning Repository available at: https://archive.ics.uci.edu/ml/datasets.php

- ## Data Preprocessing:

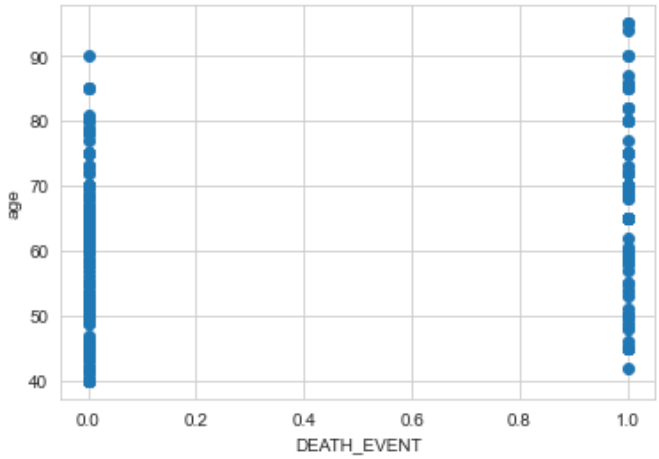
The dataset taken from the repository is complete and preprocessed already. No further preprocessing was done on dataset as it was not found in the raw form. Dataset was cleaned, integrated, reduced, and transformed by default. So, the dataset was in usable form and valid to be used for Machine Learning Algorithms.

- ## Visualization:

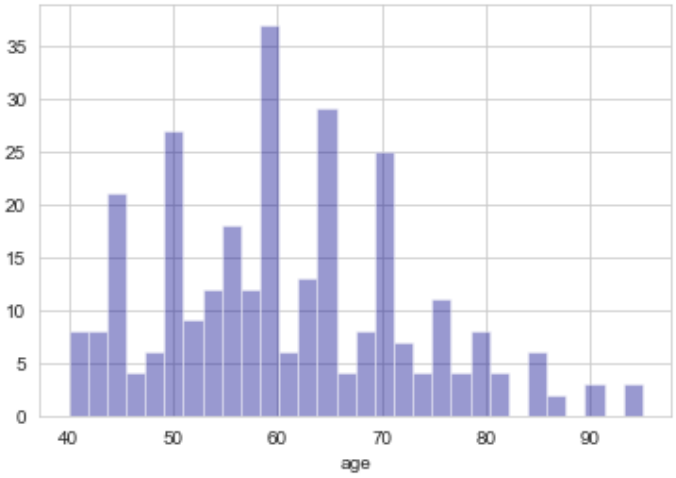
The first graph is based on the variable “DEATH_EVENT” on x-axis and the number of records as y-axis.



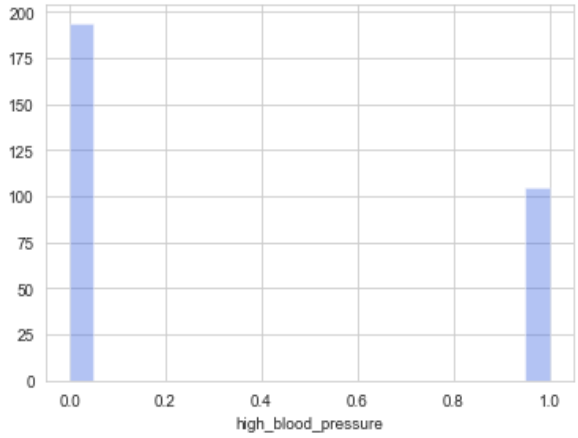
The second graph is any attribute (Age in this case) on y-axis and the DEATH_EVENT on x-axis. The output is as follows;



The third graph is of the attribute(age) on the x-axis and the records on the y-axis. The output is as follows;



The fourth graph is of the attribute(high_blood_pressure) on the x-axis and the records on the y-axis. The output is as follows;



- **Model Implementation Details:**

Introduction:

Logistic Regression is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable. In logistic regression, the dependent variable is a binary variable that contains data coded as 1 (yes, success, etc.) or 0 (no, failure, etc.). In other words, the logistic regression model predicts P(Y=1) as a function of X.

Implementation of model on dataset:

1. The code imports necessary files needed to run the algorithm on dataset such as “numpy” for linear algebra, “pandas” for data processing for csv file, etc.
2. In the second line, the csv file containing dataset is read.
3. Next, the head function is used to show the first five values from the dataset.
4. Next, the describe function is used to compute and display the statistics of the dataset.
5. In the next few lines, graphical representation of the attributes are displayed.
6. Then, the code imports train_test_split function from sklearn.model_selection which is used for splitting the dataset into two arrays:for training data and for testing data.In the next lines, the train_test_split model is imported. Here the size for testing data is also specified.
7. In the next line, X_test.head() function is used to show the first five values from the test dataset.

8. In the next line, logistic regression is imported from sklearn.linear_model and is applied to the test data with max_iterations=7600
9. LRModel.fit(X_train,y_train) function is used to fit the training data.
10. The classification report function imported performs the process of generating a report which tells about the accuracy, average, and weighted average
11. In the next lines, we have given some values to the attributes and applied the model on it for the prediction of the results.

- **Results:**

The results shows that with the given values of attributes, the model predicts that the person will have chances for heart failure.
The classification report is as follows;

	precision	recall	f1-score	support
0	0.91	0.89	0.90	45
1	0.69	0.73	0.71	15
accuracy			0.85	60
macro avg	0.80	0.81	0.80	60
weighted avg	0.85	0.85	0.85	60

- **Code:**

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
df1=pd.read_csv("heart_failure_clinical_records_dataset.csv")
df1.head()
df1.describe()
sns.set_style('whitegrid')
sns.countplot(x='DEATH_EVENT',hue='DEATH_EVENT',data=df1,palette='cubehelix')
plt.scatter(x='DEATH_EVENT',y='age',data=df1)
plt.ylabel('age')
plt.xlabel('DEATH_EVENT')
sns.distplot(df1['age'],kde=False,color='darkblue',bins=30)
sns.distplot(df1['high_blood_pressure'],kde=False,color='royalblue',bins=20)
from sklearn.model_selection import train_test_split
df1.head()
x=['age','anaemia','creatinine_phosphokinase','diabetes','ejection_fraction','high_blood_pr
essure','platelets','serum_creatinine','serum_sodium','sex','smoking','time']
y=['DEATH_EVENT']
```

```
df2=pd.DataFrame(data=df1)
df2.head()
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(df1.drop('DEATH_EVENT',axis=1),df1['D
EATH_EVENT'],test_size=0.20,random_state=101)
X_test.head()
from sklearn.linear_model import LogisticRegression
LRModel=LogisticRegression(solver='lbfgs', max_iter=7600)
LRModel.fit(X_train,y_train)
predictions_failure=LRModel.predict(X_test)
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(y_test,predictions_failure))
# paitentid_54=pd.DataFrame([1,123,126,60,0,30.1,0.349,47],columns=x)
#Defining a sample data to test the model
x=['age','anaemia','creatinine_phosphokinase','diabetes','ejection_fraction','high_blood_pr
essure','platelets','serum_creatinine','serum_sodium','sex','smoking','time']
data=[20, 0, 2239, 0, 25, 0, 33100, 1.5, 134, 0, 0, 9]
personid_00=pd.DataFrame([data],columns=x)
personid_00.head()
df1.head()
predictions_failure=LRModel.predict(personid_00)
print(predictions_failure)
```

kNN Classification and Regression:

kNN Classification

- **Dataset Description:**

The dataset is about heart attack prediction. The prediction is based on values of 0 and 1 where 0 shows death. The output i-e the DEATH_EVENT is based on the features like HIGH BLOOD PRESSURE, DIABETES, SMOKING, etc.

The output, DEATH_EVENT is the dependent variable depending on the attributes which are independent. The dataset contains real values for each attribute where the observation made falls in the range of {0, 1}.

The dataset contains 13 columns representing attributes of AGE, ANEMIA, CREATANINE, DIABETES, EJECTION_FRACTION, HIGH_BLOOD_PRESSURE, PLATELETS, SERUM_CREATANINE, SERUM_SODIUM, SEX, SMOKING, TIME, DEATH_EVENT. It has a total of 300 rows.

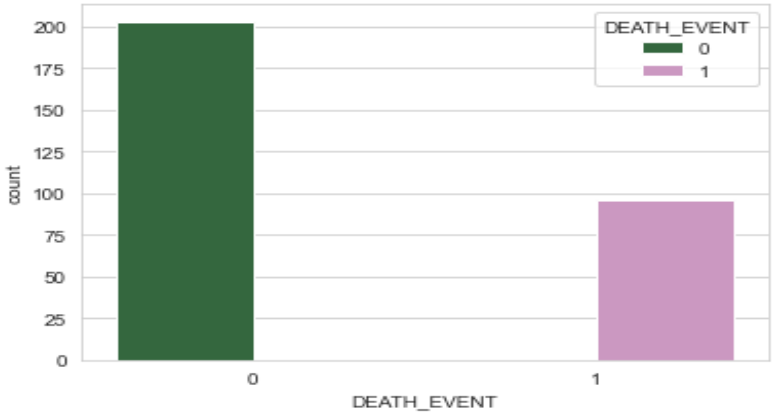
The source of the dataset is UCI Machine Learning Repository available at: https://archive.ics.uci.edu/ml/datasets.php

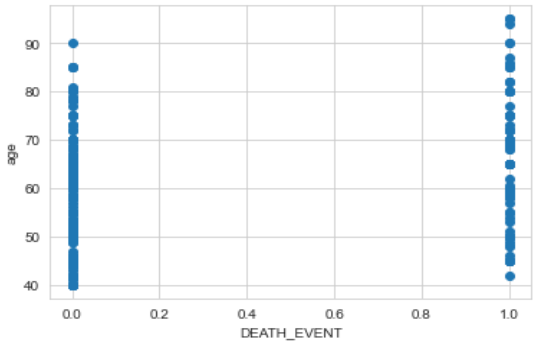
- **Data Preprocessing:**

The dataset taken from the repository is complete and preprocessed already. No further preprocessing was done on dataset as it was not found in the raw form. Dataset was cleaned, integrated, reduced, and transformed by default. So, the dataset was in usable form and valid to be used for Machine Learning Algorithms.

- **Visualization:**

The first graph is based on the variable "DEATH_EVENT" on x-axis and the number of records as y-axis. Following is the graph;





The Second graph is any attribute (Age in this case) on y-axis and the DEATH_EVENT on x-axis. The output is as follows;

- **Model Implementation Details:**

Introduction:

This algorithm is used to solve the classification model problems. K-nearest neighbor or K-NN algorithm basically creates an imaginary boundary to classify the data. When new data points come in, the algorithm will try to predict that to the nearest of the boundary line.

Therefore, larger k value means smother curves of separation resulting in less complex models. Whereas, smaller k value tends to over fit the data and resulting in complex models.

It's very important to have the right k-value when analyzing the dataset to avoid overfitting and under fitting of the dataset.

Using the k-nearest neighbor algorithm we fit the historical data (or train the model) and predict the future.

Implementation of model on dataset:

1. The code imports necessary files needed to run the algorithm on dataset such as "numpy" for linear algebra, "pandas" for data processing for csv file, etc.
2. In the second line, the csv file containing dataset is read.
3. Next, the head function is used to show the first five values from the dataset.
4. In the sixth line, the "iloc" function is used to feature the selected values and specify the location of target class which is DEATH_EVENT.
5. In the next line, the train_test_split model is imported which splits data for the purpose of training and testing data. Here the size for testing data is also specified.

6. Next, the “StandardScaler” function is imported. This function processes data so that the mean is 0 and the standard deviation is 1. The scaler.fit, scaler.transform functions fir and then transform the training data and testing data respectively so that the condition for mean and standard deviation is fulfilled.
7. Then, the kNeighborsClassifier function is imported which applies the k nearest neighbor function on training data.
8. The classification report function imported performs the process of generating a report which tells about the accuracy, average, and weighted average. The confusion matrix imported processes the grid of the actual value Vs. the predicted value.

- **Results:**

With the testing data size of 0.5 and number of neighbors for classifier, the predicted value has the accuracy of 0.8. which tells us that the results predicted from this model will be 80% accurate.

The confusion matrix for the algorithm is

Confusion Matrix:

[[106 3]
[27 14]]

And the classification report is

Classification Report:

	precision	recall	f1-score	support
0	0.80	0.97	0.88	109
1	0.82	0.34	0.48	41
accuracy			0.80	150
macro avg	0.81	0.66	0.68	150
weighted avg	0.80	0.80	0.77	150

- **Code:**

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

df1=pd.read_csv("heart_failure_clinical_records_dataset.csv")

df1.head()
```

```
X = df1.iloc[:, :-1].values #features values that we are selecting
y = df1.iloc[:, 12].values #Target class of the dataset
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.50)
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 8)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
result = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(result)
result1 = classification_report(y_test, y_pred)
print("Classification Report:",)
print (result1)
result2 = accuracy_score(y_test,y_pred)
print("Accuracy:",result2)
```

kNN Regression

- **Dataset Description:**

The dataset is about heart attack prediction. The prediction is based on values of 0 and 1 where 0 shows death. The output i-e the DEATH_EVENT is based on the features like HIGH BLOOD PRESSURE, DIABETES, SMOKING, etc.

The output, DEATH_EVENT is the dependent variable depending on the attributes which are independent. The dataset contains real values for each attribute where the observation made falls in the range of {0, 1}.

The dataset contains 13 columns representing attributes of AGE, ANEMIA, CREATANINE, DIABETES, EJECTION_FRACTION, HIGH_BLOOD_PRESSURE, PLATELETS, SERUM_CREATANINE, SERUM_SODIUM, SEX, SMOKING, TIME, DEATH_EVENT. It has a total of 300 rows.

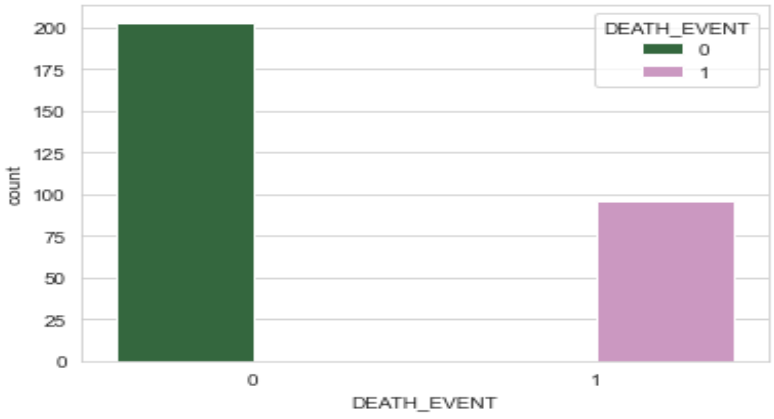
The source of the dataset is UCI Machine Learning Repository available at: https://archive.ics.uci.edu/ml/datasets.php

- **Data Preprocessing:**

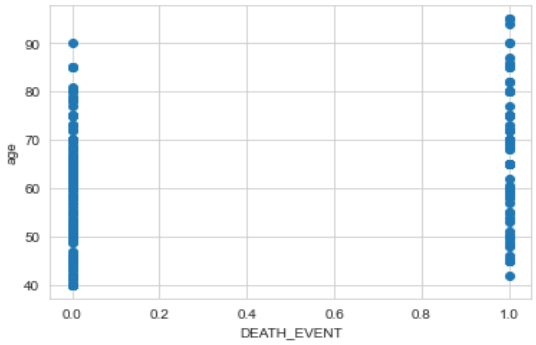
The dataset taken from the repository is complete and preprocessed already. No further preprocessing was done on dataset as it was not found in the raw form. Dataset was cleaned, integrated, reduced, and transformed by default. So, the dataset was in usable form and valid to be used for Machine Learning Algorithms.

- **Visualization:**

The first graph is based on the variable “DEATH_EVENT” on x-axis and the number of records as y-axis. Following is the graph;



The Second graph is any attribute (Age in this case) on y-axis and the DEATH_EVENT on x-axis. The output is as follows;



- **Model Implementation Details:**

Introduction:

This algorithm is used to solve the classification model problems. K-nearest neighbor or K-NN algorithm basically creates an imaginary boundary to classify the data. When new data points come in, the algorithm will try to predict that to the nearest of the boundary line.

Therefore, larger k value means smother curves of separation resulting in less complex models. Whereas, smaller k value tends to over fit the data and resulting in complex models.

It's very important to have the right k-value when analyzing the dataset to avoid overfitting and under fitting of the dataset.

Using the k-nearest neighbor algorithm we fit the historical data (or train the model) and predict the future.

Implementation of model on dataset:

1. The code imports necessary files needed to run the algorithm on dataset such as "numpy" for linear algebra, "pandas" for data processing for csv file, etc.
2. In the second line, the csv file containing dataset is read.
3. Next, the head function is used to show the first five values from the dataset.
4. In the sixth line, the "iloc" function is used to feature the selected values and specify the location of target class which is DEATH_EVENT.
5. In the next line, the train_test_split model is imported which splits data for the purpose of training and testing data. Here the size for testing data is also specified.
6. Next, the "StandardScaler" function is imported. This function processes data so that the mean is 0 and the standard deviation is 1. The scaler.fit, scaler.transform functions fir and then transform the training data and testing data respectively so that the condition for mean and standard deviation is fulfilled.

7. Then, the kNeighborsRegressor function is imported which applies the k nearest neighbor function on testing data. The closest neighbor count is taken as 8.
8. The predict function then outputs the average value from the testing dataset.

- **Results:**

The model showed the average value of the testing data.

The value is 0.75

- **Code:**

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

df1=pd.read_csv("heart_failure_clinical_records_dataset.csv")

df1.head()

X = df1.iloc[:, :1].values #features values that we are selecting

y = df1.iloc[:, 1].values

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.10)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(X_train)

X_train = scaler.transform(X_train)

X_test = scaler.transform(X_test)

from sklearn.neighbors import KNeighborsRegressor

reg = KNeighborsRegressor(n_neighbors = 8)

reg.fit(X_train, y_train)

y_pred = reg.predict(X_test) print(reg.predict([[0.2]]))
```