

Heart Failure Diseases

1) Uploading file:

In google colab to upload a file from our local drive we need to import **upload** from **google.colab.files** library.

```
#upload data file to the colab directory
from google.colab.files import upload
file = upload()
```

Choose Files heart_failur..._dataset.csv

- heart_failure_clinical_records_dataset.csv(text/csv) - 12239 bytes, last modified: 10/25/2022 - 100% done

Saving heart_failure_clinical_records_dataset.csv to heart_failure_clinical_records_dataset.csv

2) Importing Packages:

Here we have imported so many python packages and modules.

The main difference between a module and a package in Python is that a module is a simple Python script with a **.py** extension file that contains collections of functions and global variables. In contrast, a package is a directory that contains a collection of modules, and this directory also contains an **__init__.py** file by which the interpreter interprets it as a package.

The library is having a collection of related functionality of codes that allows you to perform many tasks without writing your code. **It is a reusable chunk of code that we can use by importing it into our program**, we can just use it by importing that library and calling the method of that library with a period(.). However, it is often assumed that while a package is a collection of modules, a library is a collection of packages.

Some modules are Datetime, Regex, Random etc. Some Packages are Numpy, Pandas etc. Some Libraries are Matplotlib, Seaborn etc.

Numpy: NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in NumPy is called ndarray , it provides a lot of supporting functions that make working with ndarray very easy.

Pandas: The **pandas** library in Python is used to work with dataframes which structures data in rows and columns. It is widely used in data analysis and machine learning.

Matplotlib: Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.

Sklearn: sklearn library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, and clustering and dimensionality reduction.

Keras: Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.

```
import numpy as np #for calculate data
import pandas as pd #for data frame related function use
import matplotlib.pyplot as plt #for creating static, animated, and interactive visualizations
from sklearn import preprocessing #change raw feature vectors into a representation that is more suitable for the downstream estimators
from sklearn.preprocessing import StandardScaler #helps to get standardized distribution, with a zero mean and standard deviation of one
from sklearn.model_selection import train_test_split #Split arrays or matrices into random train and test subsets.
import seaborn as sns #for making statistical graphics
from keras.layers import Dense,Dropout #represents of layers of the model
from keras.models import Sequential #for a plain stack of layers where each layer has exactly one input tensor and one output tensor.
# from tensorflow.keras.utils import to_categorical #plot categorical data matplotlib
from keras import callbacks #to customize the behavior of a Keras model during training, evaluation, or inference
from sklearn.metrics import confusion_matrix #show the report about the data
```

3) Loading DATA:

Here, we load the CSV using `pd.read_csv()` method from pandas.

The **head** function in Python displays the first five rows of the dataframe by default.

```
#loading data
data = pd.read_csv("heart_failure_clinical_records_dataset.csv")#read csv file
data.head() #show first 5 rows from the data
```

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	smoking	time	DEATH_1_YEAR
0	75.0	0	582	0	20	1	265000.00	1.9	130	1	0	4	
1	55.0	0	7861	0	38	0	263358.03	1.1	136	1	0	6	
2	65.0	0	146	0	20	0	162000.00	1.3	129	1	1	7	
3	50.0	1	111	0	20	0	210000.00	1.9	137	1	0	7	
4	65.0	1	160	1	20	0	327000.00	2.7	116	0	0	8	

4) DATA INFO:

The `info()` method prints information about the DataFrame.

The information contains the number of columns, column labels, column data types, memory usage, range index, and the number of cells in each column (non-null values).

About The Data:

- **Age:** Age of the patient
- **Anemia:** If the patient had the hemoglobin below the normal range
- **Creatinine phosphokinase:** The level of the creatinine phosphokinase in the blood in mcg/L
- **Diabetes:** If the patient was diabetic
- **ejection_fraction:** Ejection fraction is a measurement of how much blood the left ventricle pumps out with each contraction
- **High blood pressure:** If the patient had hypertension
- **Platelets:** Platelet count of blood in kilo platelets/mL
- **Serum creatinine:** The level of serum creatinine in the blood in mg/dL
- **Serum sodium:** The level of serum sodium in the blood in mEq/L
- **Sex:** The sex of the patient
- **Smoking:** If the patient smokes actively or ever did in past
- **Time:** It is the time of the patient's follow-up visit for the disease in months
- **DEATH_EVENT:** If the patient deceased during the follow-up period

```
data.info() #info about all the feature present in the dataset
```

```
<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
```

5) Data Analysis:

We begin our analysis by plotting a count plot of the target attribute. A correlation matrix of the various attributes to examine the feature importance.

Seaborn is an amazing visualization library for statistical graphics plotting in Python. It provides beautiful default styles and color palettes to make statistical plots more attractive. It is built on the top of [matplotlib](#) library and also closely integrated to the data structures from pandas.

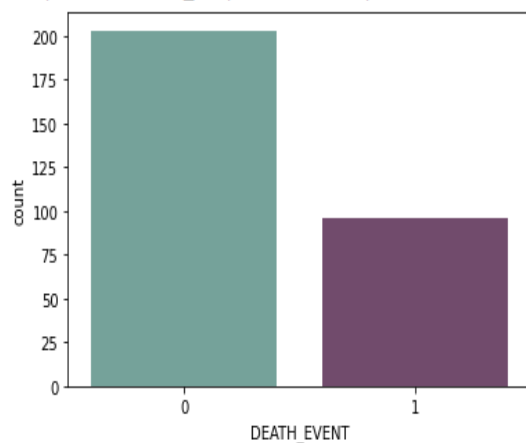
sns.countplot() method is used to Show the counts of observations in each categorical bin using bars.

- **Palette :** This parameter take palette name, list, or dict, Colors to use for the different levels of the hue variable. Should be something that can be interpreted by `color_palette()`, or a dictionary mapping hue levels to matplotlib colors.
- **x, y:** This parameter take names of variables in data or vector data, optional, Inputs for plotting long-form data.



```
#first of all let us evaluate the target and find out if our data is imbalanced or not
cols= ["#6daa9f", "#774571"] #list of colors for each column
sns.countplot(x= data["DEATH_EVENT"], palette= cols) #ploting how many death occurred and how many not death
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f418fdebc50>



6) Data Preprocessing:

To Delete a column from a Pandas DataFrame or Drop one or more than one column from a DataFrame. **Drop Columns from a Dataframe** using **drop()** method.

#assigning values to features as X and target as y

X= data.drop(["DEATH_EVENT"],axis=1) #data without DEATH_EVENT columns

y=data["DEATH_EVENT"] #data of DEATH_EVENT

X

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	smoking	time
0	75.0	0	582	0	20	1	265000.00	1.9	130	1	0	4
1	55.0	0	7861	0	38	0	263358.03	1.1	136	1	0	6
2	65.0	0	146	0	20	0	162000.00	1.3	129	1	1	7
3	50.0	1	111	0	20	0	210000.00	1.9	137	1	0	7
4	65.0	1	160	1	20	0	327000.00	2.7	116	0	0	8
...
294	62.0	0	61	1	38	1	155000.00	1.1	143	1	1	270
295	55.0	0	1820	0	38	0	270000.00	1.2	139	0	0	271
296	45.0	0	2060	1	60	0	742000.00	0.8	138	0	0	278
297	45.0	0	2413	0	38	0	140000.00	1.4	140	1	1	280
298	50.0	0	196	0	45	0	395000.00	1.6	136	1	1	285

299 rows x 12 columns

7) Set up Standard Scaler:

The Python sklearn module has a method called **StandardScaler()** which returns a Scaler object with methods for transforming data sets.

- The **fit(data)** method is used to compute the mean and std dev for a given feature to be used further for scaling.
- The **transform(data)** method is used to perform scaling using mean and std dev calculated using the .fit() method.
- The **fit_transform()** method does both fits and transform.

A Pandas **DataFrame** is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

The **describe()** method returns description of the data in the DataFrame.

If the DataFrame contains numerical data, the description contains these information for each column:

count - The number of not-empty values.
mean - The average (mean) value.
std - The standard deviation.
min - the minimum value.
25% - The 25% percentile*.
50% - The 50% percentile*.
75% - The 75% percentile*.
max - the maximum value.

*Percentile meaning: how many of the values are less than the given percentile.

The property **T** is an accessor to the method **transpose()**.

<pre>#Set up a standard scaler for the features col_names = list(X.columns) #make a list of the columns or input features s_scaler = preprocessing.StandardScaler() #create reference of StandardScaler X_df= s_scaler.fit_transform(X) #scale the all the input feature # X_df X_df = pd.DataFrame(X_df, columns=col_names) #convert 2d X_df array to dataframe X_df.describe().T #describe some statistical calculation about each feature</pre>								
	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
anaemia	299.0	1.009969e-16	1.001676	-0.871105	-0.871105	-0.871105	1.147968	1.147968
creatinine_phosphokinase	299.0	0.000000e+00	1.001676	-0.576918	-0.480393	-0.342574	0.000166	7.514640
diabetes	299.0	9.060014e-17	1.001676	-0.847579	-0.847579	-0.847579	1.179830	1.179830
ejection_fraction	299.0	-3.267546e-17	1.001676	-2.038387	-0.684180	-0.007077	0.585389	3.547716
high_blood_pressure	299.0	0.000000e+00	1.001676	-0.735688	-0.735688	-0.735688	1.359272	1.359272
✓ 0s completed at 12:49 AM								

8) Training Sets and Splitting Test:

The **train_test_split()** class from **sklearn.model_selection** is used to split our data into train and test sets where feature variables are given as input in the method.

test_size determines the portion of the data which will go into test sets and a random state is used for data reproducibility.

Inside the parenthesis, we'll provide the name of the "X" input data as the first argument. This data should contain the feature data. Optionally, we can also provide the name of the "y" dataset that contains the label or target dataset

The `random_state` parameter controls how the pseudo-random number generator randomly selects observations to go into the training set or test set.

```
#splitting test and training sets
X_train, X_test, y_train, y_test = train_test_split(X_df, y, test_size=0.25, random_state=7) #training data has 75% and test data size of 25% of data
```

9) MODEL BUILDING:

Initializing the ANN:

- 1) **callbacks.earlyStopping()** is a callback function of Keras used for stopping training when training data stop improving. This method accepts the following parameters.
 - **minDelta:** It should be a number. It is the minimum value below which is not considered an improvement in training.
 - **patience:** It should be a number. It is the number of times it should not stop when it encounters a value that is below than minDelta.
 - **restoreBestWeights:** It should be a boolean value. It tells whether to restore the best value from the monitored quantity in each epoch or not.
- 2) A **Sequential** model is appropriate for a **plain stack of layers** where each layer has **exactly one input tensor and one output tensor**. Add layers via the `.add()` method. **Dense** implements the operation: **output = activation(dot(input, kernel) + bias)** where **activation** is the element-wise activation function passed as the **activation** argument, **kernel** is a weights matrix created by the layer, and **bias** is a bias vector created by the layer (only applicable if **use_bias** is **True**). These are all attributes of **Dense**.
- 3) **Arguments**

- **units:** Positive integer, dimensionality of the output space.
- **activation:** Activation function to use. If you don't specify anything, no activation is applied (ie. "linear" activation: $a(x) = x$).
- **kernel_initializer:** Initializer for the **kernel** weights matrix
- In Keras, the input dimensions needs to be given excluding the batch size

4) **activation function**-->An Activation Function decides whether a neuron should be activated or not. This means that it will decide whether the neuron's input to the network is important or not in the process of prediction using simpler mathematical operations.

5) **relu**-->Relu is a non-linear function or piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero.
 $f(x) = \max(0, x)$

6) **sigmoid**-->Sigmoid function takes any real-valued input, and outputs a value between zero and one.

7) **kerner_initializer**-->Initializers define the way to set the initial random weights of Keras layers

```
early_stopping = callbacks.EarlyStopping(
    min_delta=0.001, # minimum amount of change to count as an improvement
    patience=30, # how many epochs to wait before stopping
    restore_best_weights=True #best weights will be chosen by the model
)

# Initialising the model type
model = Sequential()

# layers
#input layer
model.add(Dense(units = 16, kernel_initializer = 'uniform', activation = 'relu', input_dim = 12))
#hidden layer1
model.add(Dense(units = 8, kernel_initializer = 'uniform', activation = 'relu')) #has 8 nodes
model.add(Dropout(0.25)) # a technique where randomly selected(25%) neurons are ignored during tr
#hidden layer2
model.add(Dense(units = 4, kernel_initializer = 'uniform', activation = 'relu'))#has 4 nodes
model.add(Dropout(0.5)) #50% neurons are ignored during training
#output layer
model.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid')) #has only 1 n
```

10) Compiling ANN:

Optimizer: An optimizer is a function or an algorithm that modifies the attributes of the neural network, such as weights and learning rate. [Adam](#) is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data.

Loss function: A loss function is a function that compares the target and predicted output values; measures how well the neural network models the training data.

[binary_crossentropy](#) Used as a loss function for binary classification model. The [binary_crossentropy](#) function computes the cross-entropy loss between true labels and predicted labels.

Metrics: Metrics help in evaluating deep learning models. Metrics are used to monitor and measure the performance of model. [Accuracy](#) is one metric for evaluating classification models. Informally, [accuracy](#) is the fraction of predictions our model got right.

```
[10] # from tensorflow.keras.optimizers import SGD
      # Compiling the ANN
      #binary_crossentropy for our binary classification
      model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```


11) Training ANN:

batch_size: defines the number of samples that will be propagated through the network

epochs = 500:The model will train maximum 500 times


callbacks: Customize the behavior of a Keras model during training, evaluation, or inference

validation_split: When training and testing a neural net, it's important to separate training data from validation, so that you aren't checking the accuracy of the model with the same data that you use to train it. This will help reduce overfitting. We'll use the `validation_split` parameter when fitting our model to automatically split data up into a training set and a validation set, and use that to check the validation accuracy of our model.

 # Train the ANN

```
history = model.fit(X_train, y_train, batch_size = 32, epochs = 500, callbacks=[early_stopping], validation_split=0.2)
```

Epoch 50/500

 6/6 [=====] - 0s 9ms/step - loss: 0.5183 - accuracy: 0.6480 - val_loss: 0.5201 - val_accuracy: 0.6667

Epoch 37/500

6/6 [=====] - 0s 7ms/step - loss: 0.4893 - accuracy: 0.6480 - val_loss: 0.5200 - val_accuracy: 0.6667

Epoch 38/500

6/6 [=====] - 0s 8ms/step - loss: 0.5121 - accuracy: 0.6480 - val_loss: 0.5199 - val_accuracy: 0.6667

Epoch 39/500

6/6 [=====] - 0s 8ms/step - loss: 0.5123 - accuracy: 0.6480 - val_loss: 0.5188 - val_accuracy: 0.6667

numpy.mean(arr, axis = None) : Compute the arithmetic mean (average) of the given data (array elements) along the specified axis.

```
[12] val_accuracy = np.mean(history.history['val_accuracy']) # mean value of all epochs accuracy
print("\n%s: %.2f%%" % ('val_accuracy', val_accuracy*100)) #print in percentage formation
```

val_accuracy: 67.79%


Keras **model predicts** is the method of function provided in Keras that helps in the predictions of output depending on the specified samples of input to the model.

 # Predicting the test set results

```
y_pred = model.predict(X_test) #predict with test data set
```

```
y_pred = (y_pred > 0.5) #make predicted value to 0 or 1:
```

```
y_pred
```

 [False],
[False],
[False],
[False],
[False],
_ _ _

```

▶ y_test
268    0
240    0
278    0
176    0
202    0
..
24     1
62     0
249    0
90     0
50     1
Name: DEATH_EVENT, Length: 75, dtype: int64

```

A **confusion matrix** is a table that is often used to **describe the performance of a classification model** (or "classifier") on a set of test data for which the true values are known. The confusion matrix itself is relatively simple to understand, but the related terminology can be confusing.

Heatmap is defined as a graphical representation of data using colors to visualize the value of the matrix. In this, to represent more common values or higher activities brighter colors basically reddish colors are used and to represent less common or activity values, darker colors are preferred. Heatmap is also defined by the name of the shading matrix. Heatmaps in Seaborn can be plotted by using the **seaborn.heatmap()** function.

In a classification problem, the summary of the prediction results is stored inside a confusion matrix. We have to plot the confusion matrix to look at the count of correct and incorrect predictions. To plot a confusion matrix, we have to create a data frame of the confusion matrix, and then we can use the **heatmap()** function of Seaborn to plot the confusion matrix in Python.

