NAME: Vishal Shashikant Salvi **UID: 2019230069**

NAME: Shivam Pawar **UID: 2019230068**

NAME: Shreyas Patel **UID: 2018130043**

CLASS: TE COMPS **BATCH: C**

EXPERIMENT No 8

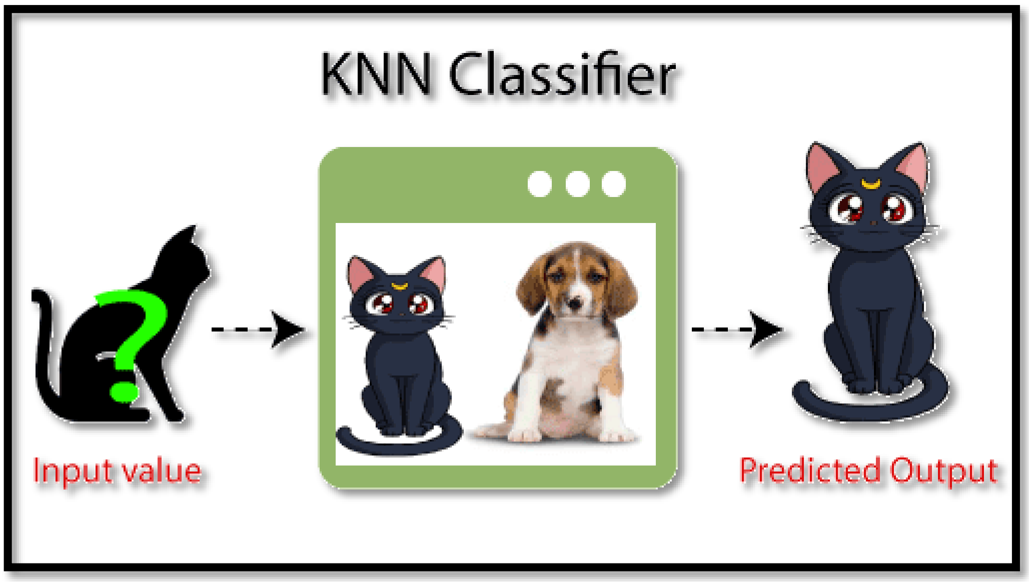
Aim: Classification using KNN

Objective:

Theory:

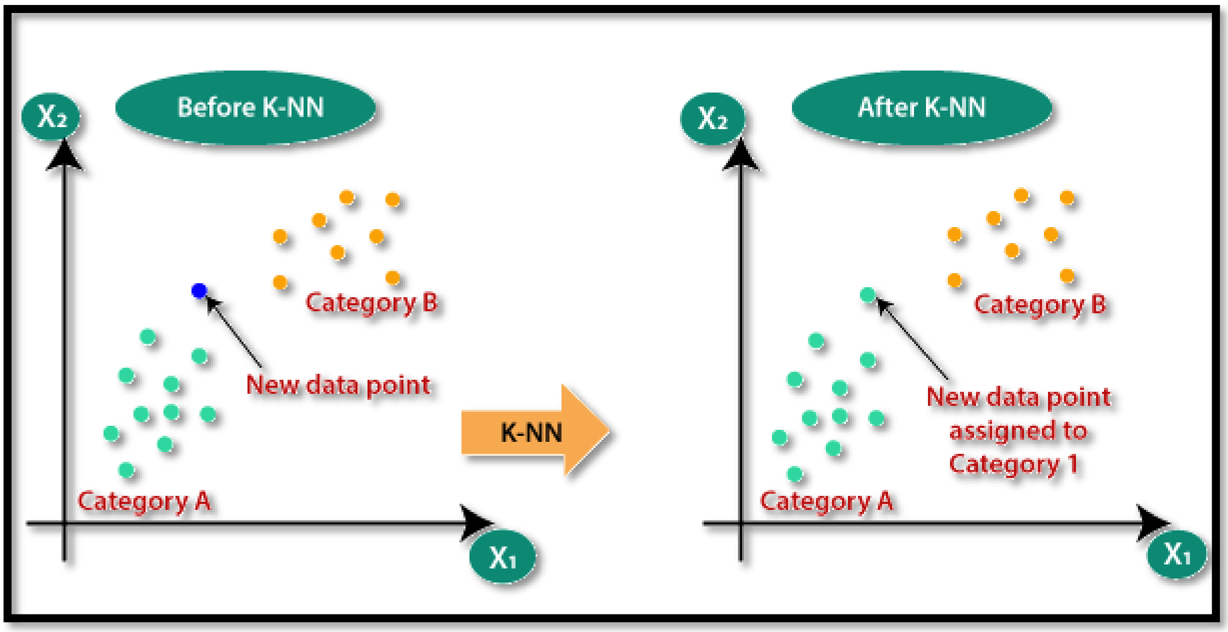
K-Nearest Neighbor(KNN) Algorithm for Machine Learning

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.
- **Example:** Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.



Why do we need a K-NN Algorithm?

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:



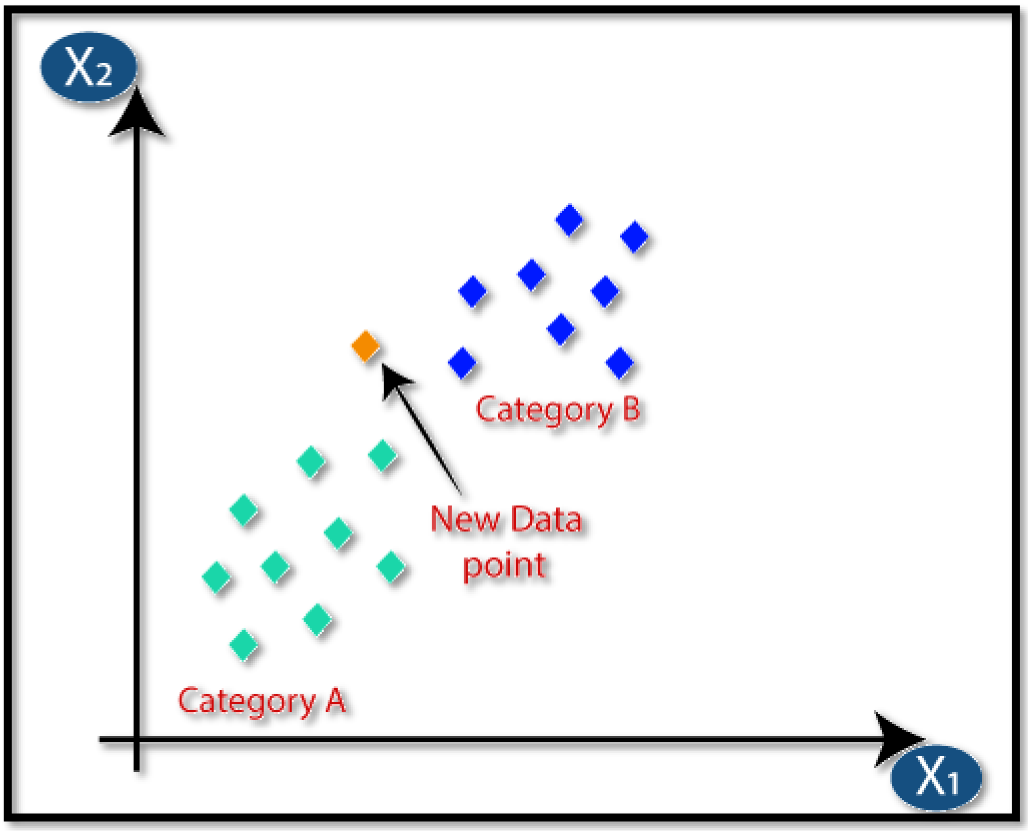
How does K-NN work?

The K-NN working can be explained on the basis of the below algorithm:

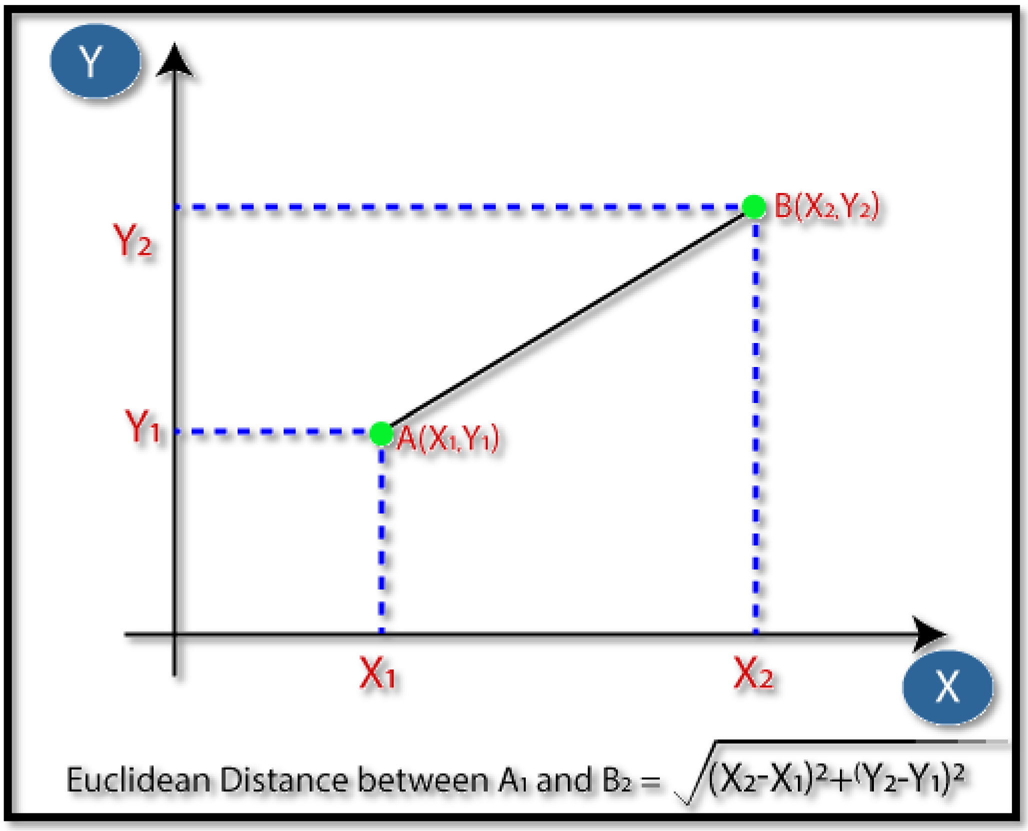
- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**

- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

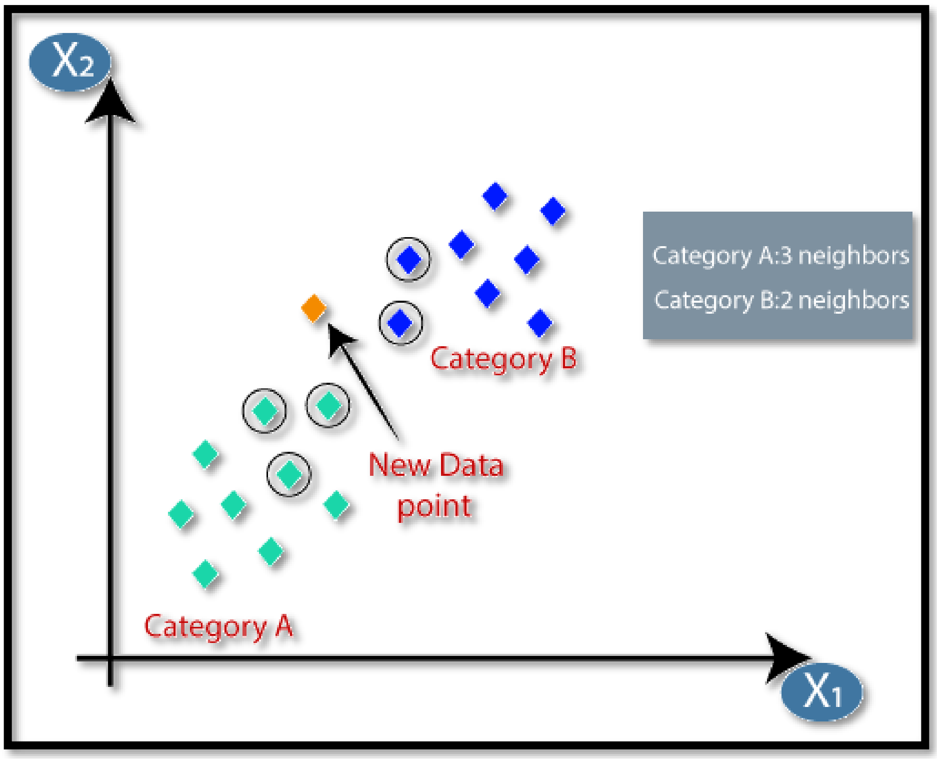
Suppose we have a new data point and we need to put it in the required category. Consider the below image:



- Firstly, we will choose the number of neighbors, so we will choose the k=5.
- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



- By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:



- As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

How to select the value of K in the K-NN Algorithm?

Below are some points to remember while selecting the value of K in the K-NN algorithm:

- There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.

Advantages of KNN Algorithm:

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.

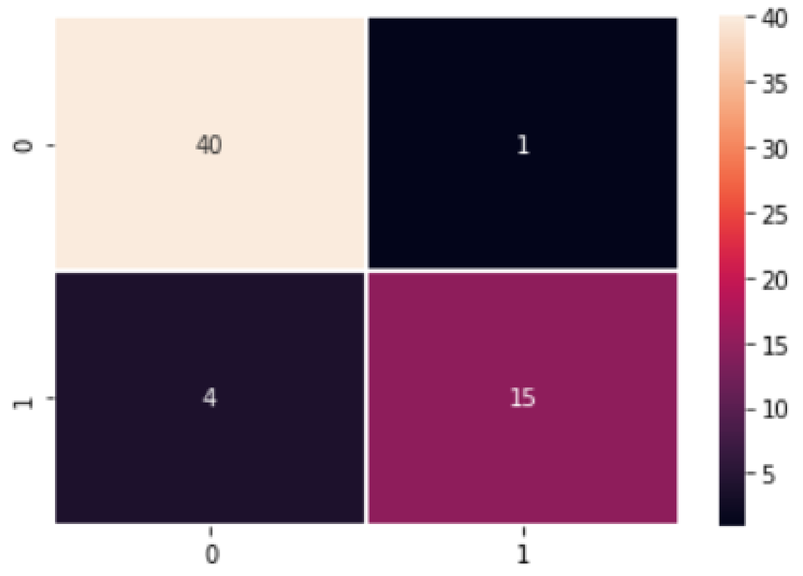
Disadvantages of KNN Algorithm:

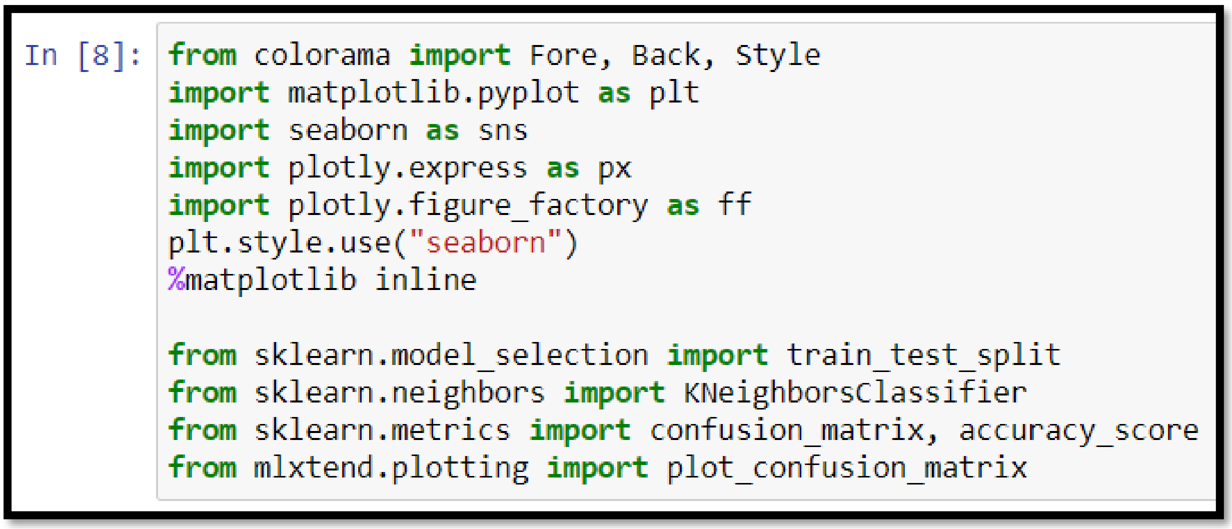
- Always needs to determine the value of K which may be complex some time.
- The computation cost is high because of calculating the distance between the data points for all the training samples.

Implementation:

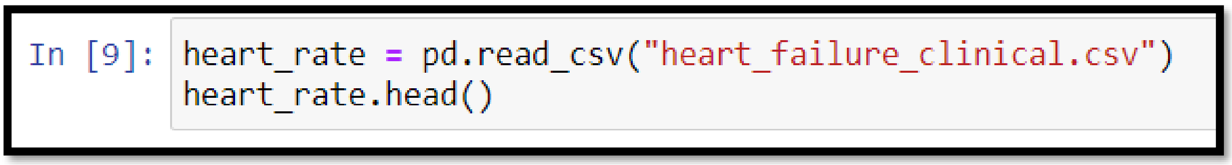
```
In [166]: #knn
          knn = KNeighborsClassifier()
          param_grid = {'n_neighbors': np.arange(1, 25)}
          knn_gscv = GridSearchCV(knn, param_grid, cv=4)
          knn_gscv.fit(X_train, Y_train)
          print("Best K Value is ",knn_gscv.best_params_)

          accuracy_list.append(knn_gscv.score(X_test,Y_test))
          print("test accuracy ",knn_gscv.score(X_test,Y_test))
          algorithm.append("K Nearest Neighbors Classifier")

          cm = confusion_matrix(Y_test,knn_gscv.predict(X_test))
          predict_list.append(cm.item(0)+cm.item(2))
          sns.heatmap(cm,annot=True, linewidths=.5)
          plt.show()
```

```
Best K Value is  {'n_neighbors': 9}
test accuracy  0.9166666666666666
```

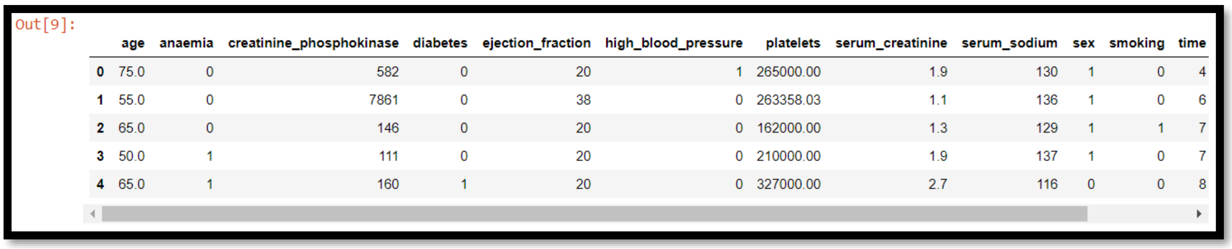




```
In [8]: from colorama import Fore, Back, Style
        import matplotlib.pyplot as plt
        import seaborn as sns
        import plotly.express as px
        import plotly.figure_factory as ff
        plt.style.use("seaborn")
        %matplotlib inline

        from sklearn.model_selection import train_test_split
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.metrics import confusion_matrix, accuracy_score
        from mlxtend.plotting import plot_confusion_matrix
```

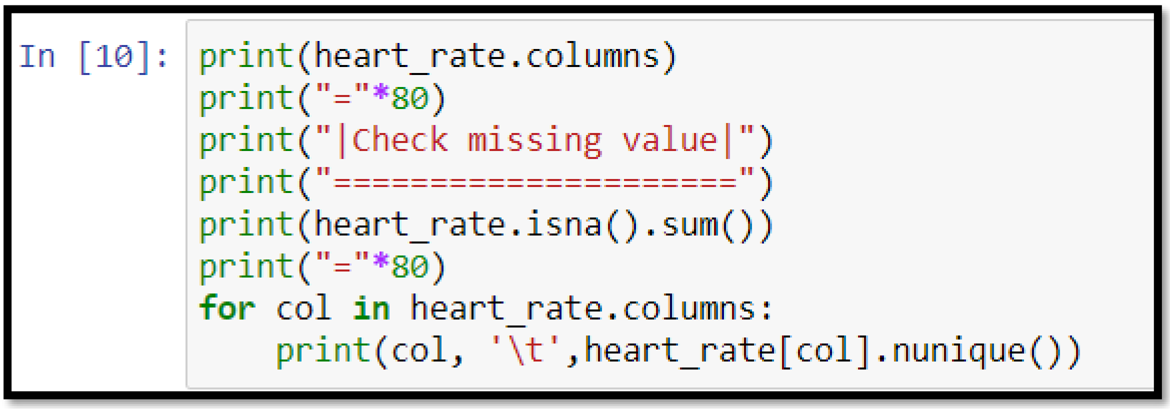


```
In [9]: heart_rate = pd.read_csv("heart_failure_clinical.csv")
        heart_rate.head()
```

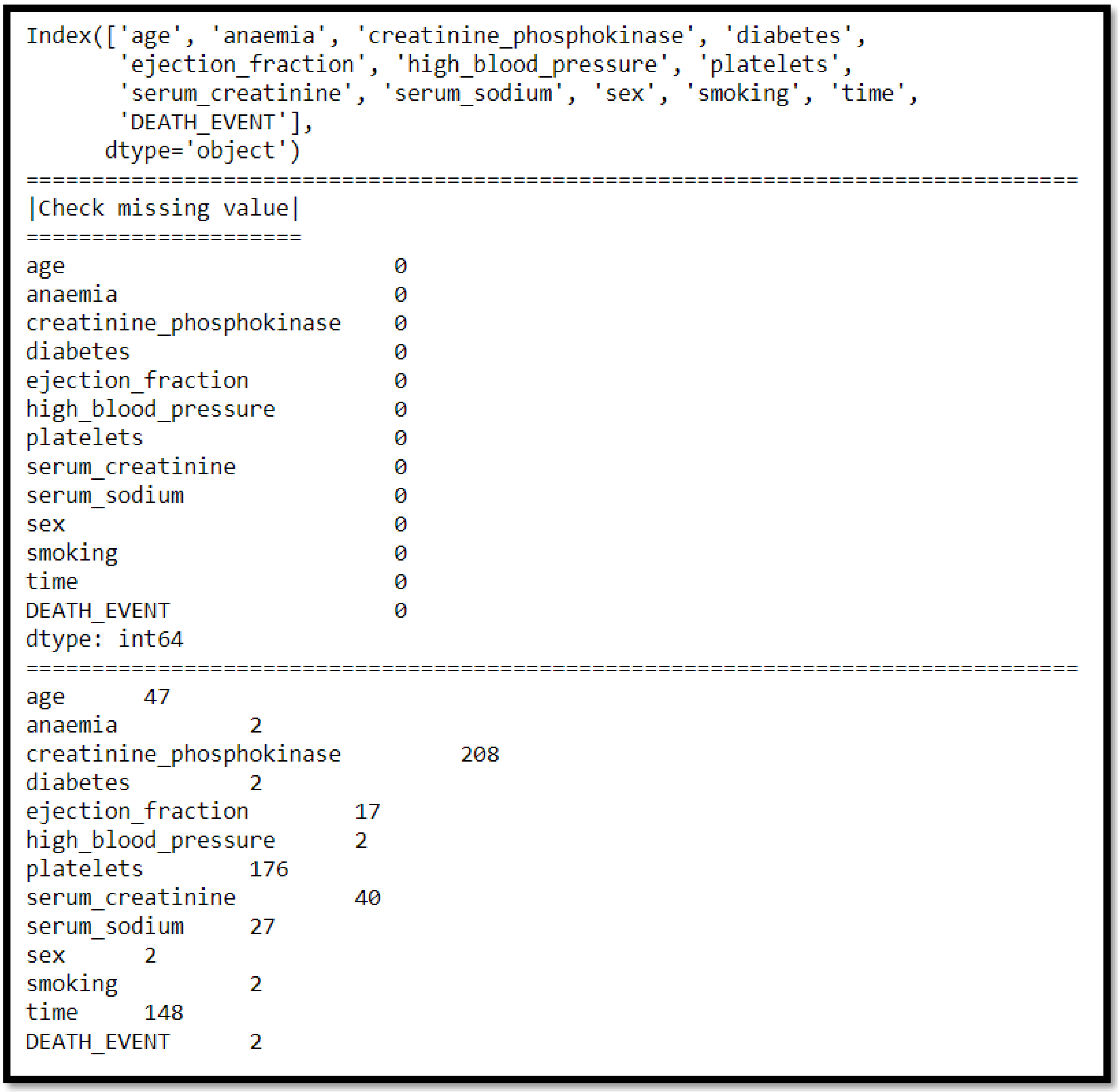


Out[9]:

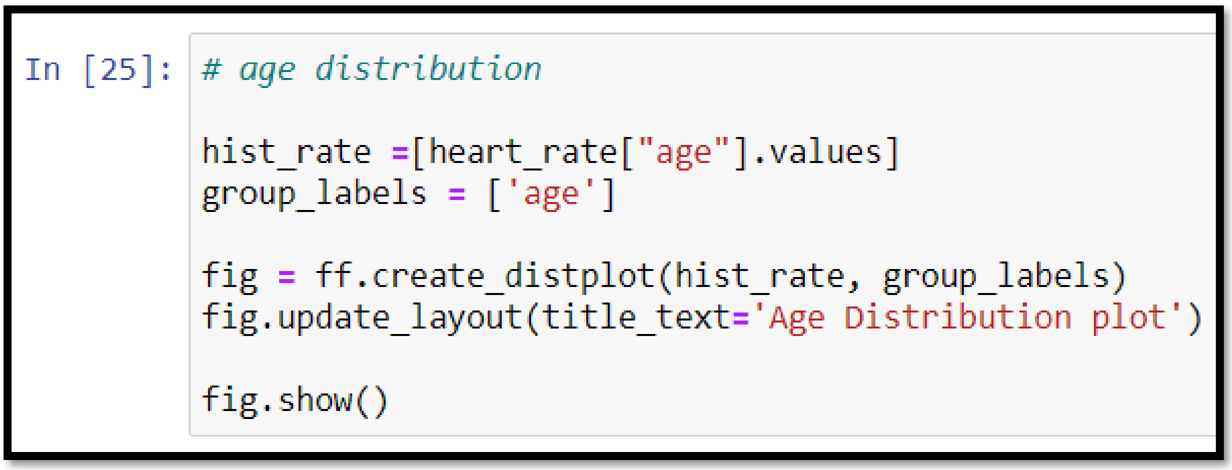
	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	smoking	time
0	75.0	0	582	0	20	1	265000.00	1.9	130	1	0	4
1	55.0	0	7861	0	38	0	263358.03	1.1	136	1	0	6
2	65.0	0	146	0	20	0	162000.00	1.3	129	1	1	7
3	50.0	1	111	0	20	0	210000.00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



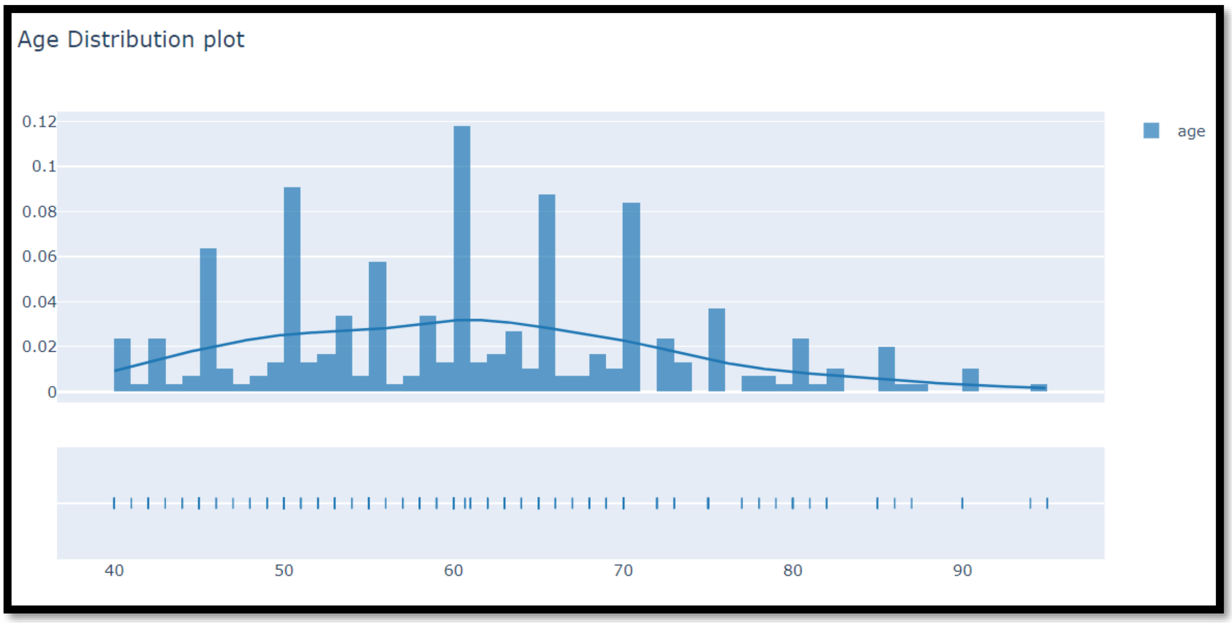
```
In [10]: print(heart_rate.columns)
         print("="*80)
         print("|Check missing value|")
         print("=====================")
         print(heart_rate.isna().sum())
         print("="*80)
         for col in heart_rate.columns:
             print(col, '\t',heart_rate[col].nunique())
```

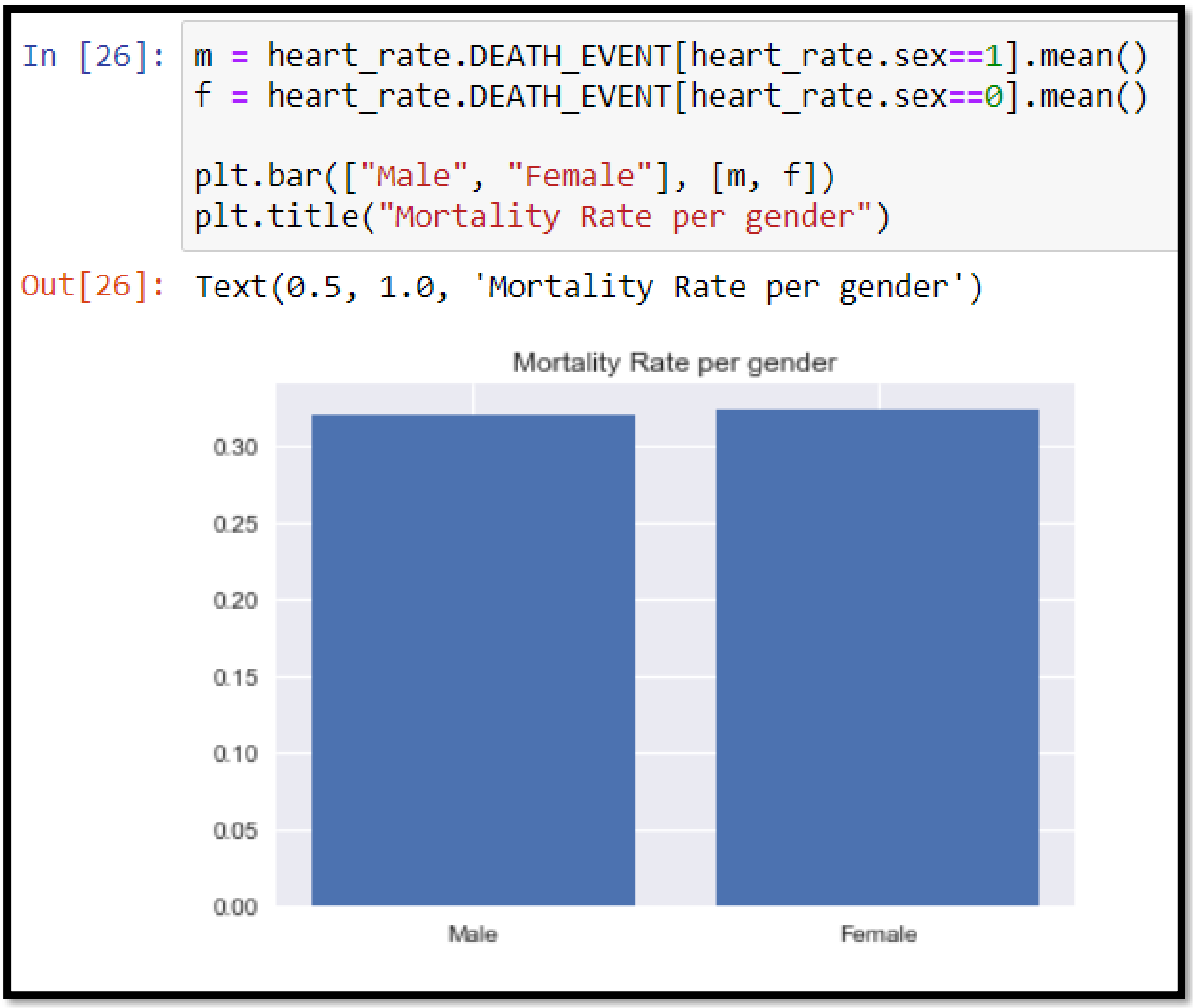


```
Index(['age', 'anaemia', 'creatinine_phosphokinase', 'diabetes',
       'ejection_fraction', 'high_blood_pressure', 'platelets',
       'serum_creatinine', 'serum_sodium', 'sex', 'smoking', 'time',
       'DEATH_EVENT'],
      dtype='object')
================================================================================
|Check missing value|
=====================
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
================================================================================
age      47
anaemia          2
creatinine_phosphokinase         208
diabetes         2
ejection_fraction        17
high_blood_pressure      2
platelets        176
serum_creatinine         40
serum_sodium     27
sex      2
smoking          2
time     148
DEATH_EVENT      2
```

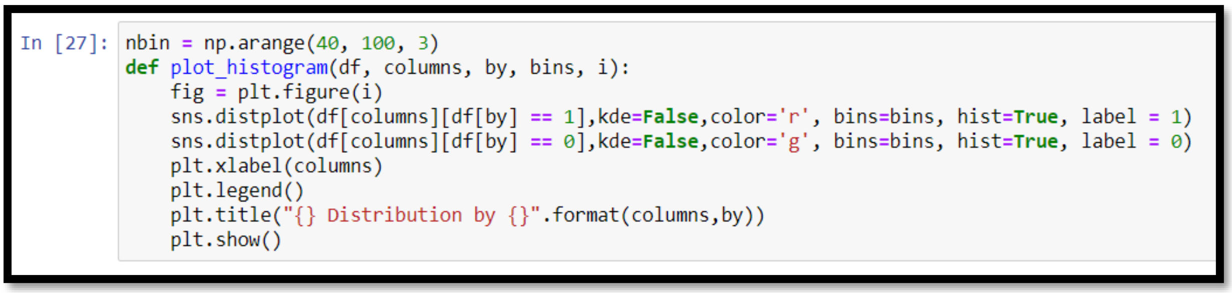


```
In [25]: # age distribution

         hist_rate =[heart_rate["age"].values]
         group_labels = ['age']

         fig = ff.create_distplot(hist_rate, group_labels)
         fig.update_layout(title_text='Age Distribution plot')

         fig.show()
```

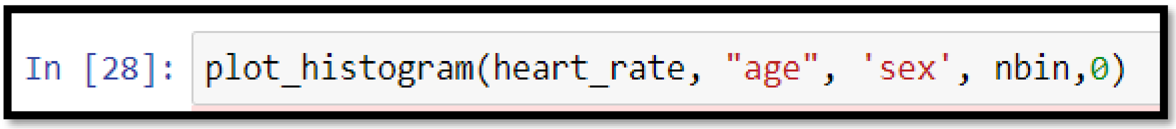




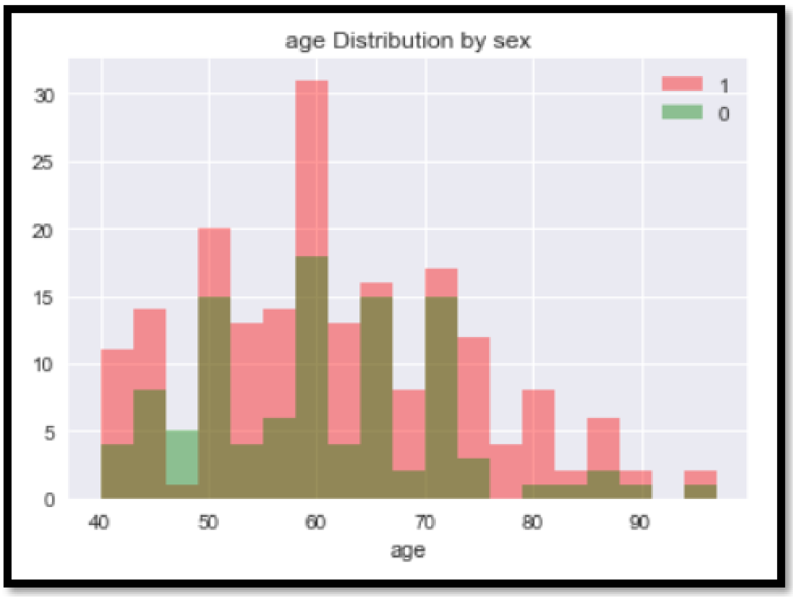
```
In [26]: m = heart_rate.DEATH_EVENT[heart_rate.sex==1].mean()
         f = heart_rate.DEATH_EVENT[heart_rate.sex==0].mean()

         plt.bar(["Male", "Female"], [m, f])
         plt.title("Mortality Rate per gender")

Out[26]: Text(0.5, 1.0, 'Mortality Rate per gender')
```

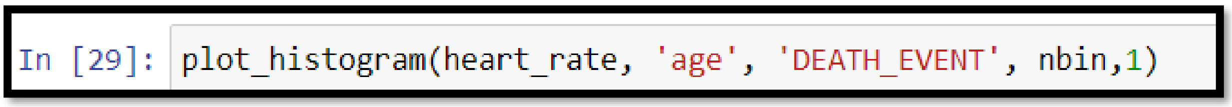


```
In [27]: nbin = np.arange(40, 100, 3)
         def plot_histogram(df, columns, by, bins, i):
             fig = plt.figure(i)
             sns.distplot(df[columns][df[by] == 1],kde=False,color='r', bins=bins, hist=True, label = 1)
             sns.distplot(df[columns][df[by] == 0],kde=False,color='g', bins=bins, hist=True, label = 0)
             plt.xlabel(columns)
             plt.legend()
             plt.title("{} Distribution by {}".format(columns,by))
             plt.show()
```

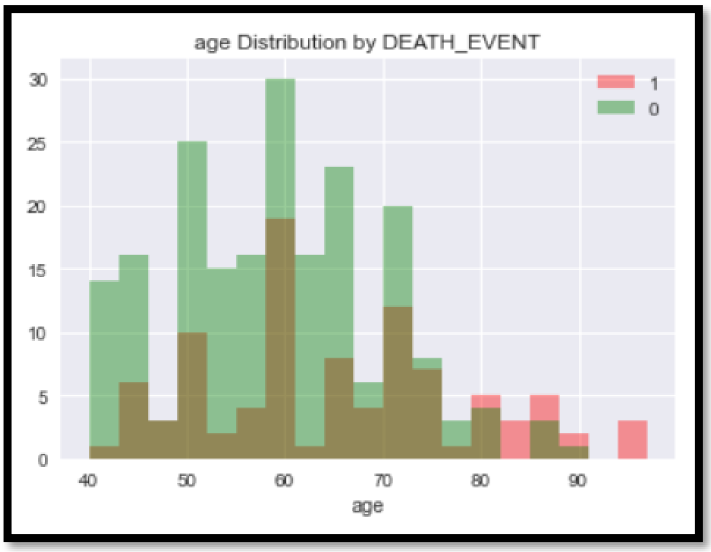


```
In [28]: plot_histogram(heart_rate, "age", 'sex', nbin,0)
```



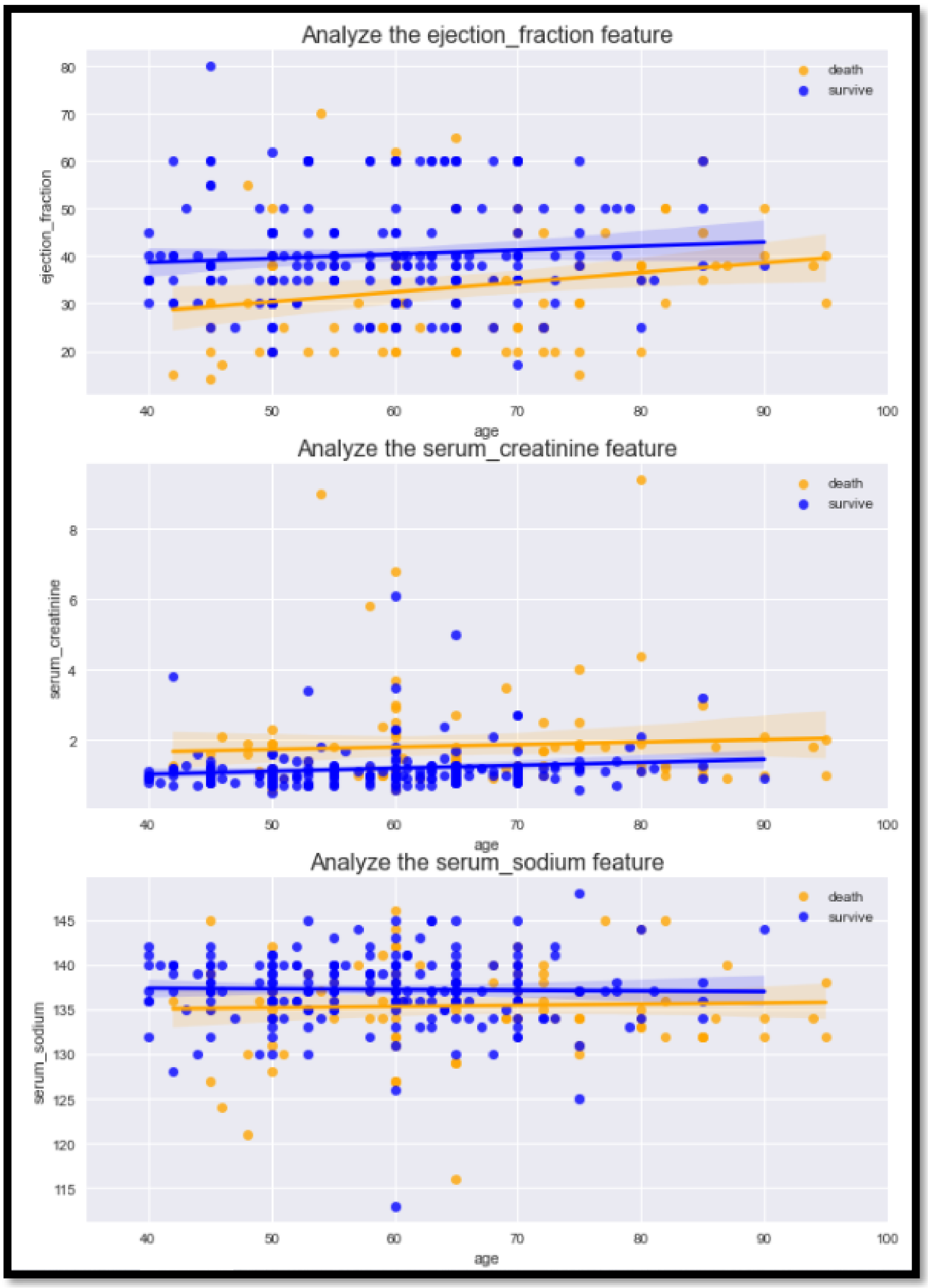


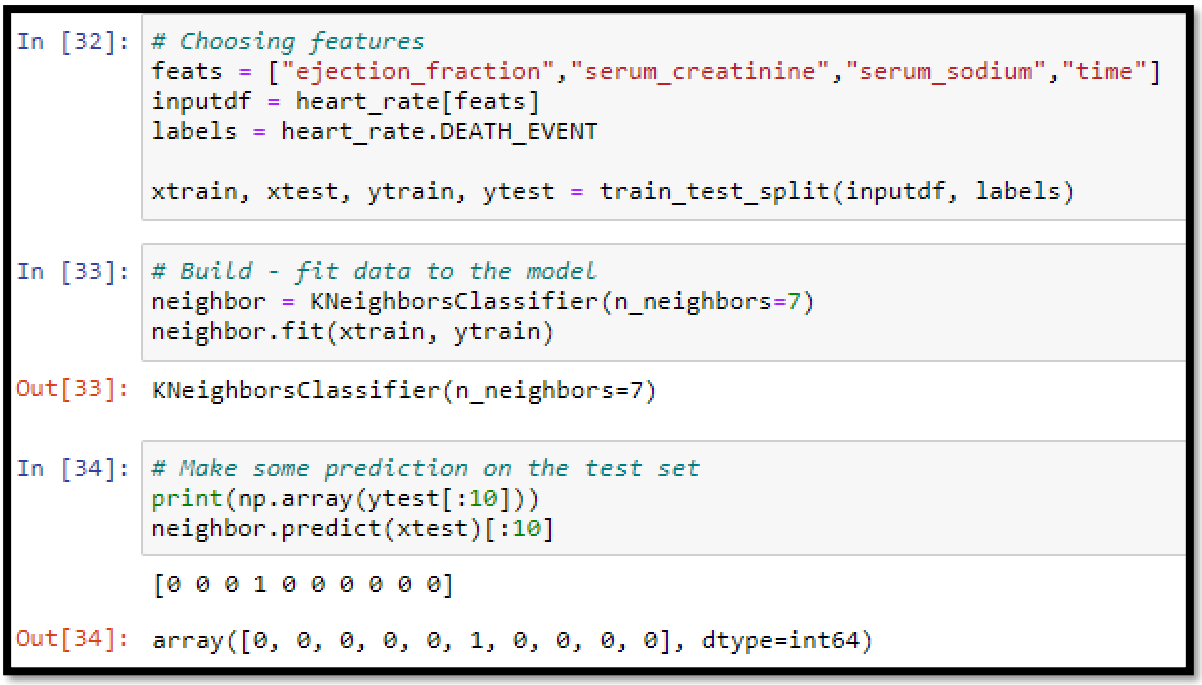
```
In [29]: plot_histogram(heart_rate, 'age', 'DEATH_EVENT', nbin,1)
```



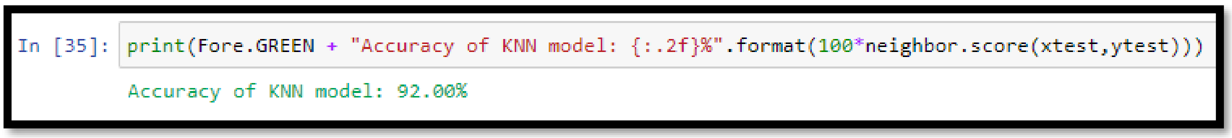
```
In [37]: def my_regplot(ax, x, y, data, hue):
             sns.regplot(x=x, y=y, data=data[data[hue]==1][[x,y]], ax=ax, color='orange', label="death")
             sns.regplot(x=x, y=y, data=data[data[hue]==0][[x,y]], ax=ax, color='blue', label='survive')
             ax.set_xlim(35,100)
             ax.set_title("Analyze the {} feature".format(y), fontsize=16)
             ax.legend()
```

```
In [38]: fig, (ax1, ax2, ax3) = plt.subplots(3, figsize=(10,15))
         # sns.lineplot(x='age', y='ejection_fraction',data=heart_rate, color='r', ax=ax1, hue="DEATH_EVENT")
         # sns.lineplot(x='age', y='serum_creatinine',data=heart_rate, color='r', ax=ax2, hue="DEATH_EVENT")
         # sns.lineplot(x='age', y='',data=heart_rate, color='r', ax=ax3, hue="DEATH_EVENT")
         my_regplot(ax1, 'age', 'ejection_fraction', heart_rate, "DEATH_EVENT")
         my_regplot(ax2, 'age', 'serum_creatinine', heart_rate, "DEATH_EVENT")
         my_regplot(ax3, 'age', 'serum_sodium', heart_rate, "DEATH_EVENT")
```

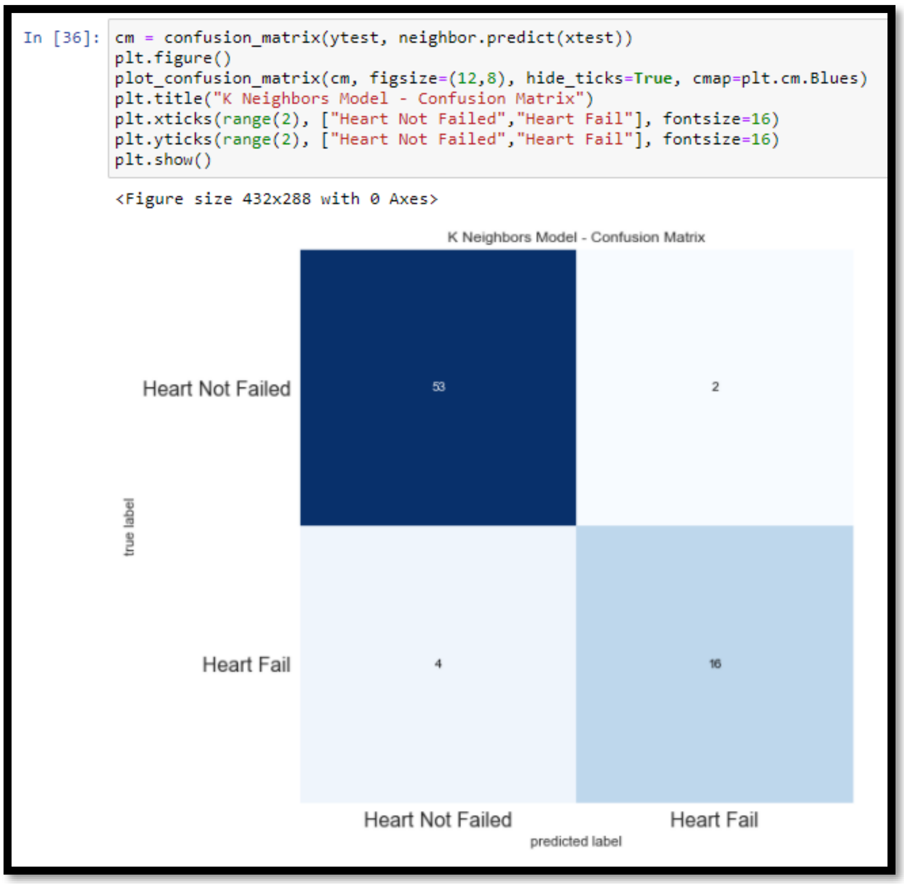




```
In [32]: # Choosing features
         feats = ["ejection_fraction","serum_creatinine","serum_sodium","time"]
         inputdf = heart_rate[feats]
         labels = heart_rate.DEATH_EVENT

         xtrain, xtest, ytrain, ytest = train_test_split(inputdf, labels)

In [33]: # Build - fit data to the model
         neighbor = KNeighborsClassifier(n_neighbors=7)
         neighbor.fit(xtrain, ytrain)

Out[33]: KNeighborsClassifier(n_neighbors=7)

In [34]: # Make some prediction on the test set
         print(np.array(ytest[:10]))
         neighbor.predict(xtest)[:10]

         [0 0 0 1 0 0 0 0 0 0]

Out[34]: array([0, 0, 0, 0, 0, 1, 0, 0, 0, 0], dtype=int64)
```



```
In [35]: print(Fore.GREEN + "Accuracy of KNN model: {:.2f}%".format(100*neighbor.score(xtest,ytest)))

         Accuracy of KNN model: 92.00%
```



```
In [36]: cm = confusion_matrix(ytest, neighbor.predict(xtest))
         plt.figure()
         plot_confusion_matrix(cm, figsize=(12,8), hide_ticks=True, cmap=plt.cm.Blues)
         plt.title("K Neighbors Model - Confusion Matrix")
         plt.xticks(range(2), ["Heart Not Failed","Heart Fail"], fontsize=16)
         plt.yticks(range(2), ["Heart Not Failed","Heart Fail"], fontsize=16)
         plt.show()

         <Figure size 432x288 with 0 Axes>
```

Conclusion:

K-Nearest neighbour classification is a general technique to learn classification based on instance and do not have to develop an abstract model from the training data set.

References:

1) https://www.edureka.co/blog/classification-in-machine-learning/
2) https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning