Heart Failure Prediction

```
#import libraries:
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objs as go
import plotly.express as px
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn import preprocessing
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier

# Importing the dataset:
heart_data=pd.read_csv("heart_failure_clinical_records_dataset(1).csv)

# To Display the data set:
heart_data
```

Visualization:

 # To visualize the dataset into graph

```
heart_data.hist(figsize=(15,15),edgecolor='black');
```

```
# Checking for null values
heart_data.isnull().sum()
```

Pie charts & Heat map:

 # Now create a pie chart for Death and Diabetes Analysis

Analysis on Diabetes:

```
import plotly.graph_objs as go
labels = ['No Diabetes','Diabetes']
diabetes_yes = heart_data[heart_data['diabetes'] ==1]
diabetes_no = heart_data[heart_data['diabetes']==0]
values = [len(diabetes_no), len(diabetes_yes)]
fig = go.Figure(data=[go.Pie(labels=labels, values=values,
hole=.2)])
fig.update_layout(
    title_text="Analysis on Diabetes")
fig.show()
```

Death Analysis:

```
import plotly.express as px

fig=px.pie(heart_data,values='diabetes',names='DEATH_EVENT',title='
Death Analysis')

fig.show()
```

Gender Vs DEATH_EVENT:

```
import plotly.graph_objects as go

from plotly.subplots import make_subplots


d1 = heart_data[(heart_data["DEATH_EVENT"]==0) &
(heart_data["sex"]==1)]

d2 = heart_data[(heart_data["DEATH_EVENT"]==1) &
(heart_data["sex"]==1)]

d3 = heart_data[(heart_data["DEATH_EVENT"]==0) &
(heart_data["sex"]==0)]

d4 = heart_data[(heart_data["DEATH_EVENT"]==1) &
(heart_data["sex"]==0)]


label1 = ["Male","Female"]

label2 = ['Male - Survived','Male - Died', "Female -  Survived",
"Female - Died"]
```

values1 = [(len(d1)+len(d2)), (len(d3)+len(d4))]

values2 = [len(d1),len(d2),len(d3),len(d4)]

 # Create subplots: use 'domain' type for Pie subplot

fig = make_subplots(rows=1, cols=2, specs=[[{'type':'domain'}, {'type':'domain'}]])

fig.add_trace(go.Pie(labels=label1, values=values1, name="GENDER"),

1, 1)

fig.add_trace(go.Pie(labels=label2, values=values2, name="GENDER VS DEATH_EVENT"),

1, 2)

 # Use `hole` to create a donut-like pie chart

fig.update_traces(hole=.4, hoverinfo="label+percent")

fig.update_layout(

title_text="GENDER DISTRIBUTION IN THE DATASET \

GENDER VS DEATH_EVENT",

 # Add annotations in the center of the donut pies.

annotations=[dict(text='GENDER', x=0.19, y=0.5, font_size=10, showarrow=False),

```
                dict(text='GENDER VS DEATH_EVENT', x=0.84,
y=0.5, font_size=9, showarrow=False)],
    autosize=False,width=1200, height=500,
paper_bgcolor="pink")


fig.show()
```

Age VS Death_count:

```
import plotly.express as px
fig = px.histogram(heart_data, x="age", color="DEATH_EVENT",
marginal="violin", hover_data=heart_data.columns,
                title ="Distribution of AGE Vs DEATH_EVENT",
                labels={"age": "AGE"},
                template="plotly_dark",
                color_discrete_map={"0": "RebeccaPurple", "1":
"MediumPurple"}
               )
fig.show()
```

Heat Map:

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(10,10))
sns.heatmap(heart_data.corr(),vmin=-1, cmap='coolwarm',annot=True);
```

DATA ANALYSIS:

#To analyze the data of DEATH_EVENT

```
cols= ["#6daa9f","#774571"]
sns.countplot(x= heart_data["DEATH_EVENT"], palette= cols)
```

Data Modeling & Algorithms:

Logistic Regression:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,accuracy_score

Feature=['time','ejection_fraction','serum_creatinine']
x=heart_data[Feature]
y=heart_data["DEATH_EVENT"]

xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.2,random_state
=2)

from sklearn.linear_model import LogisticRegression
```

```
log_re=LogisticRegression()

log_re.fit(xtrain,ytrain)
log_re_pred=log_re.predict(xtest)

log_re.fit(xtrain,ytrain)
log_re_pred=log_re.predict(xtest)

log_acc=accuracy_score(ytest,log_re_pred)
print("Logistic Accuracy Score: ","{:.2f}%".format(100*log_acc))

#Install mlxtend
%pip install mlxtend
from mlxtend.plotting import plot_confusion_matrix

#Logistic Regression-Confusion Matrix
cm = confusion_matrix(ytest, log_re_pred)
plt.figure()
plot_confusion_matrix(cm, figsize=(12,8), hide_ticks=True,
cmap=plt.cm.Blues)
plt.title("Logistic Regerssion  - 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()
```

Preprocessing:

```
# Defining independent and dependent attributes in training and test sets
X=heart_data.drop(["DEATH_EVENT"],axis=1)
y=heart_data["DEATH_EVENT"]

 #Setting up a standard scaler for the features and from sklearn import preprocessing
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
col_names = list(X.columns)
s_scaler = preprocessing.StandardScaler()
X_df= s_scaler.fit_transform(X)
X_df = pd.DataFrame(X_df, columns=col_names)
X_df.describe().T
```

Seaborn:

```
#Plotting the scaled features using boxen plots
#Import the seaborn library
import seaborn as sns
colours =["#774571","#b398af","#f1f1f1" ,"#afcdc7", "#6daa9f"]
```

```
plt.figure(figsize=(20,10))
sns.boxenplot(data = X_df,palette = colours)
plt.xticks(rotation=90)
plt.show()
```

Train/Test Split:

```
#spliting variables and training and test sets
x = heart_data.drop("DEATH_EVENT", axis = 1)
y = heart_data['DEATH_EVENT']

x_train, x_test, y_train, y_test = train_test_split(x, y, random_state =100, stratify=y, test_size = 0.3)
print(y_train.value_counts())
```

Decision Tree Classifier:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, plot_confusion_matrix
from sklearn.tree import DecisionTreeClassifier
list1 = []
for leaves in range(2,10):
    classifier = DecisionTreeClassifier(max_leaf_nodes = leaves, random_state=0, criterion='entropy')
    classifier.fit(x_train, y_train)
    y_pred = classifier.predict(x_test)
    list1.append(accuracy_score(y_test,y_pred)*100)
print("Decision Tree Classifier Top 5 Success Rates:")
```

```
print([round(i, 2) for i in sorted(list1, reverse=True)[:5]])
plot_confusion_matrix(classifier, x_test, y_test)
plt.show()
```

K Nearest Neighbors:

```
from sklearn.metrics import accuracy_score, f1_score,confusion_matrix, recall_score, precision_score, classification_report
from sklearn.model_selection import cross_val_score, cross_val_predict
KNN = KNeighborsClassifier(n_neighbors=8)
KNN.fit(x_train, y_train)

y_test_pred_KNN = KNN.predict(x_test)
y_train_pred_KNN = KNN.predict(x_train)

test_acc_KNN = accuracy_score(y_test, y_test_pred_KNN)
train_acc_KNN = accuracy_score(y_train, y_train_pred_KNN)
scores_KNN = cross_val_score(KNN, x_train , y_train , cv = 10, scoring = 'accuracy' )

precision_score_KNN = precision_score(y_test, y_test_pred_KNN)
recall_score_KNN = recall_score(y_test, y_test_pred_KNN)
f1_score_KNN = f1_score(y_test, y_test_pred_KNN)
conf_KNN = confusion_matrix(y_test, y_test_pred_KNN)
```

```
accuracy_score_KNN = accuracy_score(y_test, y_test_pred_KNN)

print("accuracy score:", accuracy_score_KNN)
print("Train set Accuracy: ", train_acc_KNN)
print("Test set Accuracy: ", test_acc_KNN)
print("cv:  %s\n"% scores_KNN.mean())
print("***********************************************")
print("precision_score: ", precision_score_KNN)
print("recall_score: ", recall_score_KNN)
print("f1_score: ", f1_score_KNN)
print("***********************************************")
print("\nReport:\n%s\n"%classification_report(y_test, y_test_pred_KNN))
```

#Decision Tree Classifier

```
print(f'Decision Tree Classifier: {round(sorted(list1,
reverse=True)[0])}%')
```

#Logistic Regression

```
print(f'Logistic Regression: {round(100*log_acc, 2)} %')
```

#K Nearest Neighbors

```
print(f'K Nearest Neighbors: {(accuracy_score_KNN)}  %')
```