health-care-analytics-mini-project

October 25, 2023

[1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

#

HEART FAILURE PREDICTION

Cardiovascular diseases (CVDs) are the number 1 cause of death globally, taking an estimated 17.9 million lives each year, which accounts for 31% of all deaths worlwide. Heart failure is a common event caused by CVDs and this dataset contains 12 features that can be used to predict mortality by heart failure.

Most cardiovascular diseases can be prevented by addressing behavioural risk factors such as tobacco use, unhealthy diet and obesity, physical inactivity and harmful use of alcohol using population-wide strategies.

People with cardiovascular disease or who are at high cardiovascular risk (due to the presence of one or more risk factors such as hypertension, diabetes, hyperlipidaemia or already established disease) need early detection and management wherein a machine learning model can be of great help.

TABLE OF CONTENTS

IMPORTING LIBRARIES

[2]:
```
import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

from sklearn import svm
from keras.layers import Dense, BatchNormalization, Dropout, LSTM
from keras.models import Sequential
from keras import callbacks
from sklearn.metrics import precision_score, recall_score, confusion_matrix,␣
 ↪classification_report, accuracy_score, f1_score
```

LOADING DATA

[4]:
```
#loading data
data_df = pd.read_csv("/content/heart_failure_clinical_records_dataset.csv")
data_df.head()
```

[4]:
```
    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
```

About the data (Description of attributes)

- **age:** Age of the patient
- **anaemia:** Haemoglobin level of patient (Boolean)
- **creatinine_phosphokinase:** Level of the CPK enzyme in the blood (mcg/L)
- **diabetes:** If the patient has diabetes (Boolean)
- **ejection_fraction:** Percentage of blood leaving the heart at each contraction

- **high_blood_pressure:** If the patient has hypertension (Boolean)
- **platelets:** Platelet count of blood (kiloplatelets/mL)
- **serum_creatinine:** Level of serum creatinine in the blood (mg/dL)
- **serum_sodium:** Level of serum sodium in the blood (mEq/L)
- **sex:** Sex of the patient
- **smoking:** If the patient smokes or not (Boolean)
- **time:** Follow-up period (days)
- **DEATH_EVENT:** If the patient deceased during the follow-up period (Boolean)

[Attributes having Boolean values: 0 = Negative (No); 1 = Positive (Yes)]

DATA ANALYSIS

[5]:
```
# Checking for any missing values across the dataset
data_df.info()
```

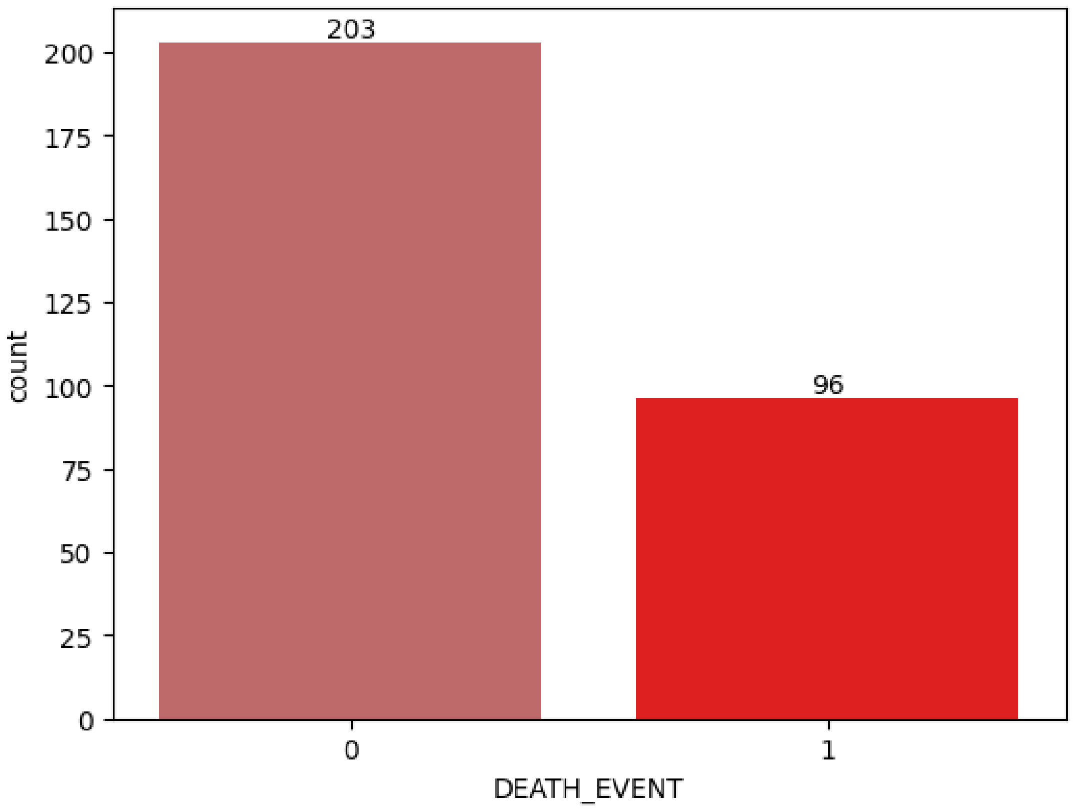
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 299 entries, 0 to 298
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   age                       299 non-null    float64
 1   anaemia              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

0.0.9 Note:

- There are 299 non-null values in all the attributes thus no missing values.
- Datatype is also either 'float64' or 'int64' which works well while feeded to an algorithm.

[6]:
```
#Evaluating the target and finding out the potential skewness in the data
cols= ["#CD5C5C","#FF0000"]
ax = sns.countplot(x= data_df["DEATH_EVENT"], palette= cols)
ax.bar_label(ax.containers[0])
```

[6]:
```
[Text(0, 0, '203'), Text(0, 0, '96')]
```



0.0.10 Note:

- Target labels are 203 versus 96 thus there is an imbalance in the data.

[7]:
```
# Doing Univariate Analysis for statistical description and understanding of␣
 ↪dispersion of data
data_df.describe().T
```

[7]:
```
                          count           mean           std      min  \
age                       299.0      60.833893     11.894809     40.0
anaemia                   299.0       0.431438      0.496107      0.0
creatinine_phosphokinase  299.0     581.839465    970.287881     23.0
diabetes                  299.0       0.418060      0.494067      0.0
ejection_fraction         299.0      38.083612     11.834841     14.0
high_blood_pressure       299.0       0.351171      0.478136      0.0
platelets                 299.0  263358.029264  97804.236869  25100.0
serum_creatinine          299.0       1.393880      1.034510      0.5
serum_sodium              299.0     136.625418      4.412477    113.0
sex                       299.0       0.648829      0.478136      0.0
smoking                   299.0       0.321070      0.467670      0.0
time                      299.0     130.260870     77.614208      4.0
```

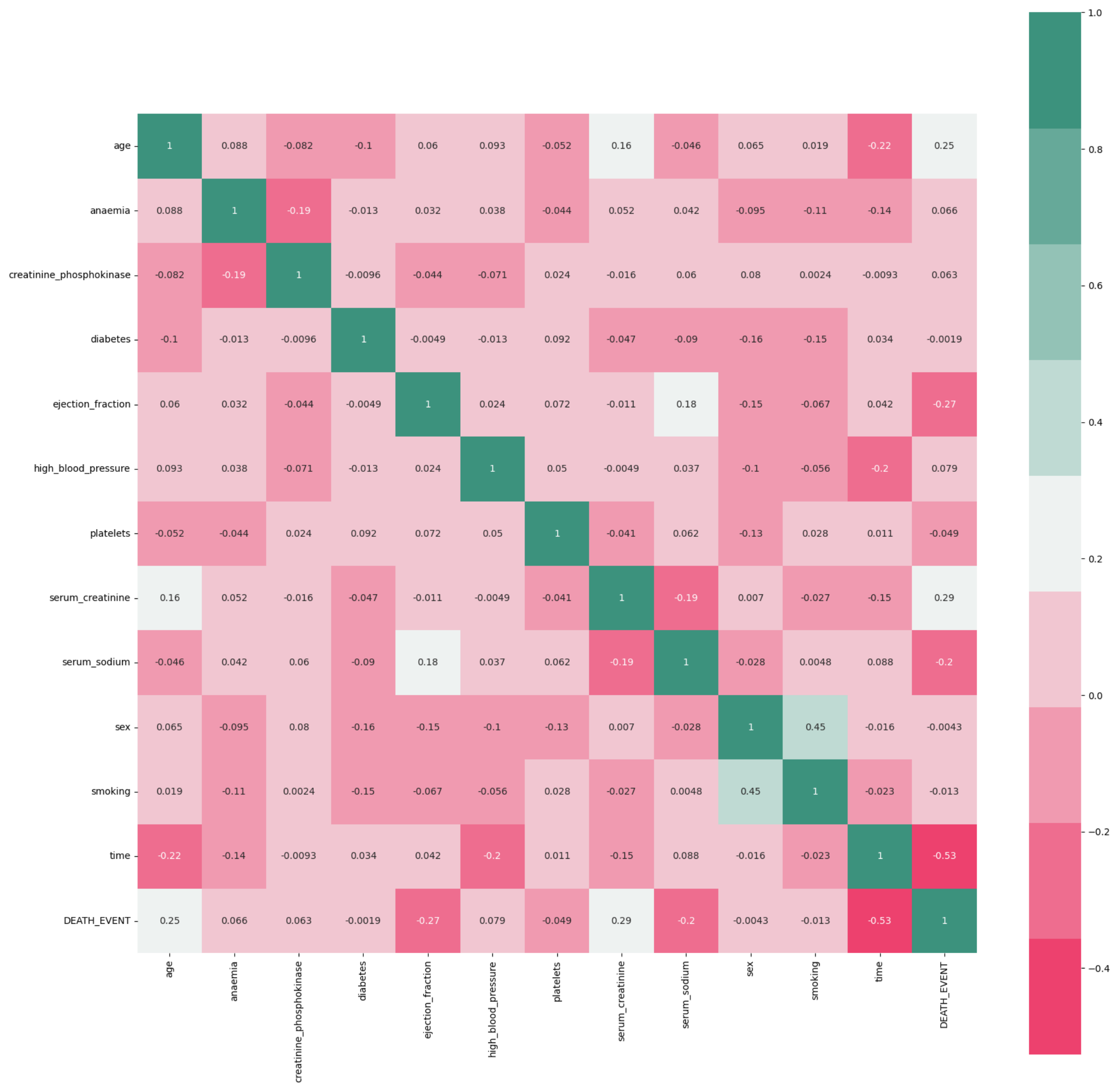
```
DEATH_EVENT                299.0       0.321070      0.467670      0.0

                               25%        50%        75%        max
age                           51.0       60.0       70.0       95.0
anaemia                        0.0        0.0        1.0        1.0
creatinine_phosphokinase     116.5      250.0      582.0     7861.0
diabetes                       0.0        0.0        1.0        1.0
ejection_fraction             30.0       38.0       45.0       80.0
high_blood_pressure            0.0        0.0        1.0        1.0
platelets                 212500.0   262000.0   303500.0   850000.0
serum_creatinine               0.9        1.1        1.4        9.4
serum_sodium                 134.0      137.0      140.0      148.0
sex                            0.0        1.0        1.0        1.0
smoking                        0.0        0.0        1.0        1.0
time                          73.0      115.0      203.0      285.0
DEATH_EVENT                    0.0        0.0        1.0        1.0
```

0.0.11 Note:

- Features “creatinine_phosphokinase” & “serum creatinine” are significantly skewed.
- All the other features almost shows the normal distribution, since mean is equal to their respective medians.

[8]:
```
#Doing Bivariate Analysis by examaning a corelation matrix of all the features␣
↪using heatmap
cmap = sns.diverging_palette(2, 165, s=80, l=55, n=9)
corrmat = data_df.corr()
plt.subplots(figsize=(20,20))
sns.heatmap(corrmat,cmap= cmap,annot=True, square=True)
```

[8]:
```
<Axes: >
```

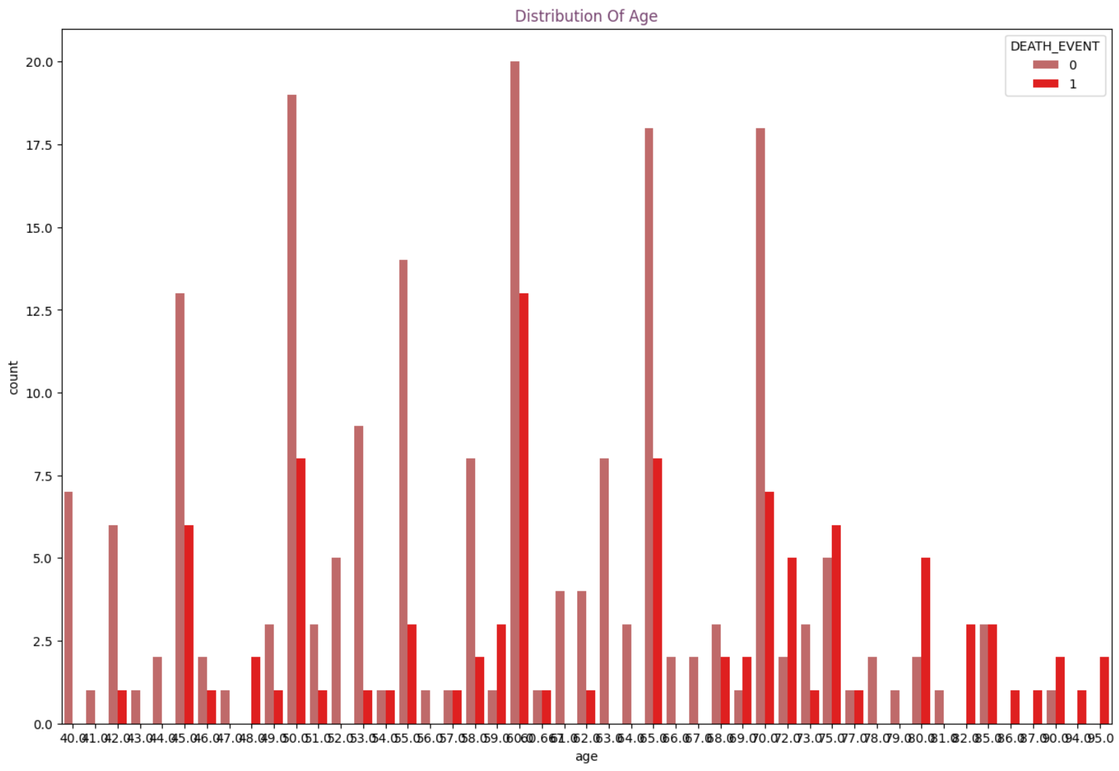


0.0.12 Note:

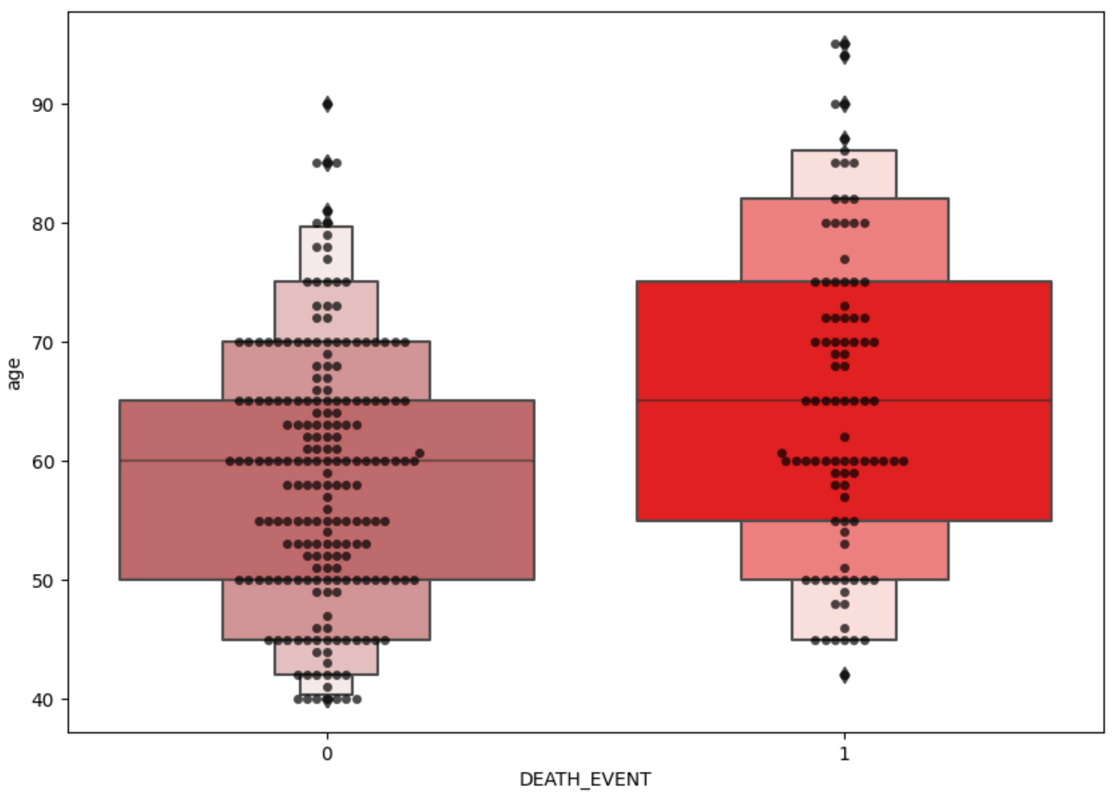
- "time" is the most important feature as it would've been very crucial to get diagnosed early with cardivascular issue so as to get timely treatment thus, reducing the chances of any fatality. (Evident from the inverse relationship)
- "serum_creatinine" is the next important feature as serum's (essential component of blood) abundancy in blood makes it easier for heart to function.
- "ejection_fraction" has also significant influence on target variable which is expected since it is basically the efficiency of the heart.
- Can be seen from the inverse relation pattern that heart's functioning declines with ageing.

```
[9]: #Evauating age distribution as per the deaths happened
     plt.figure(figsize=(15,10))
     Days_of_week=sns.countplot(x=data_df['age'],data=data_df, hue␣
      ↪="DEATH_EVENT",palette = cols)
     Days_of_week.set_title("Distribution Of Age", color="#774571")
```

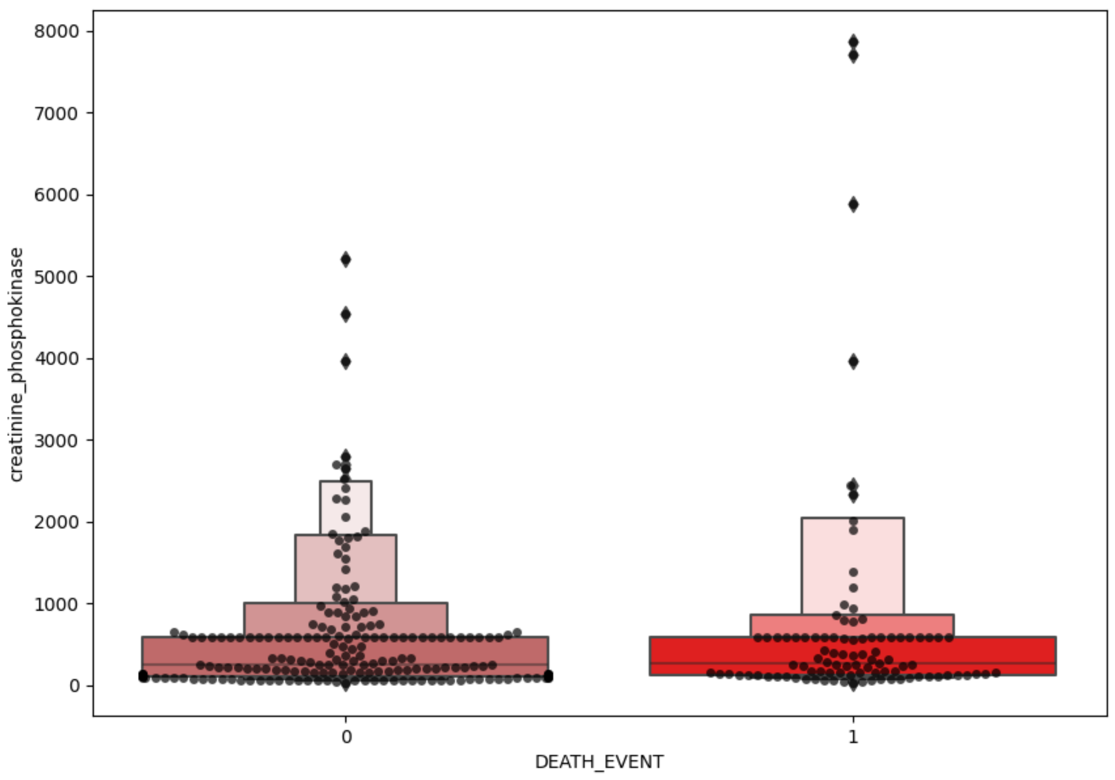
```
[9]: Text(0.5, 1.0, 'Distribution Of Age')
```



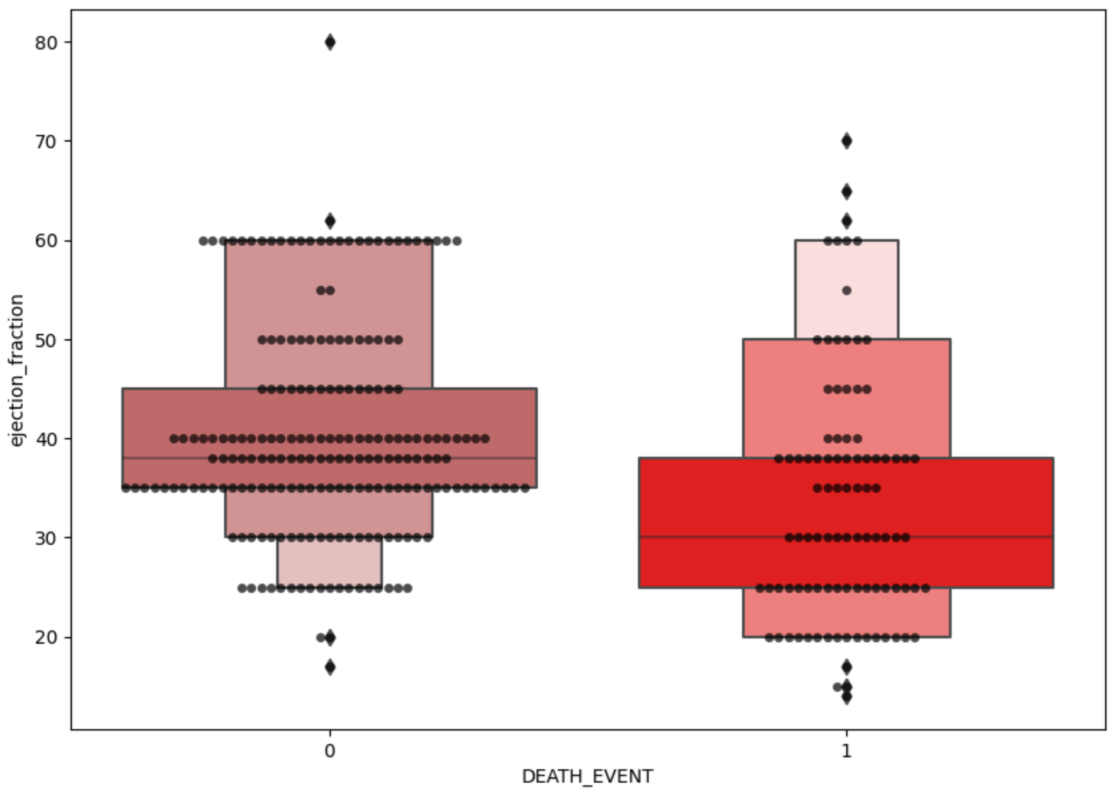
```
[10]: # Checking for potential outliers using the "Boxen and Swarm plots" of non␣
       ↪binary features.
      feature =␣
       ↪["age","creatinine_phosphokinase","ejection_fraction","platelets","serum_creatinine","serum
       ↪"time"]
      for i in feature:
          plt.figure(figsize=(10,7))
          sns.swarmplot(x=data_df["DEATH_EVENT"], y=data_df[i], color="black",␣
       ↪alpha=0.7)
          sns.boxenplot(x=data_df["DEATH_EVENT"], y=data_df[i], palette=cols)
          plt.show()
```

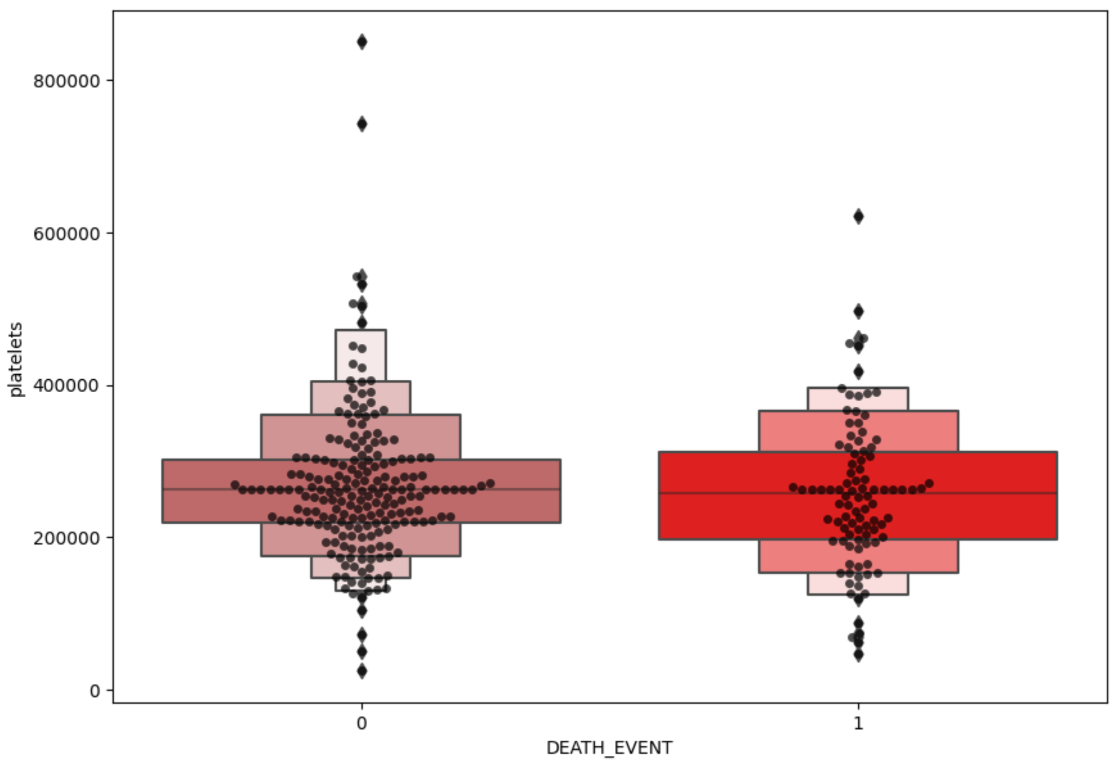


age
90
80
70
60
50
40
0
1
DEATH_EVENT

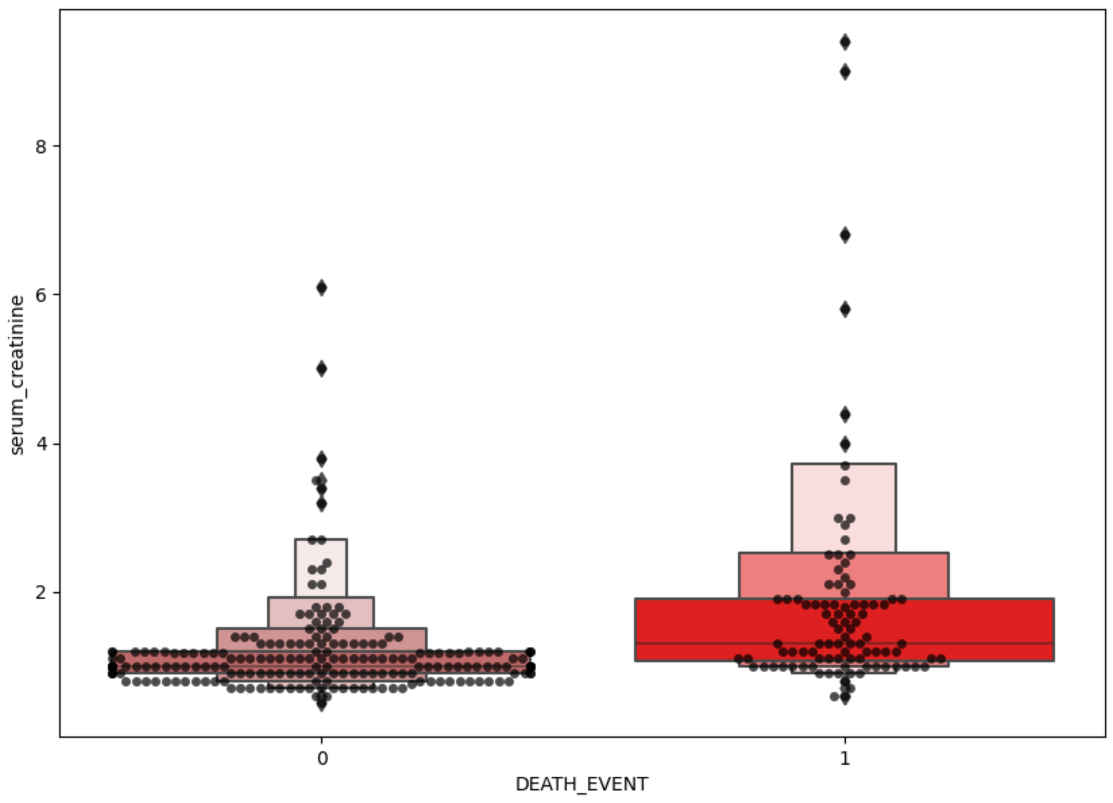


creatinine_phosphokinase
8000
7000
6000
5000
4000
3000
2000
1000
0
0
1
DEATH_EVENT

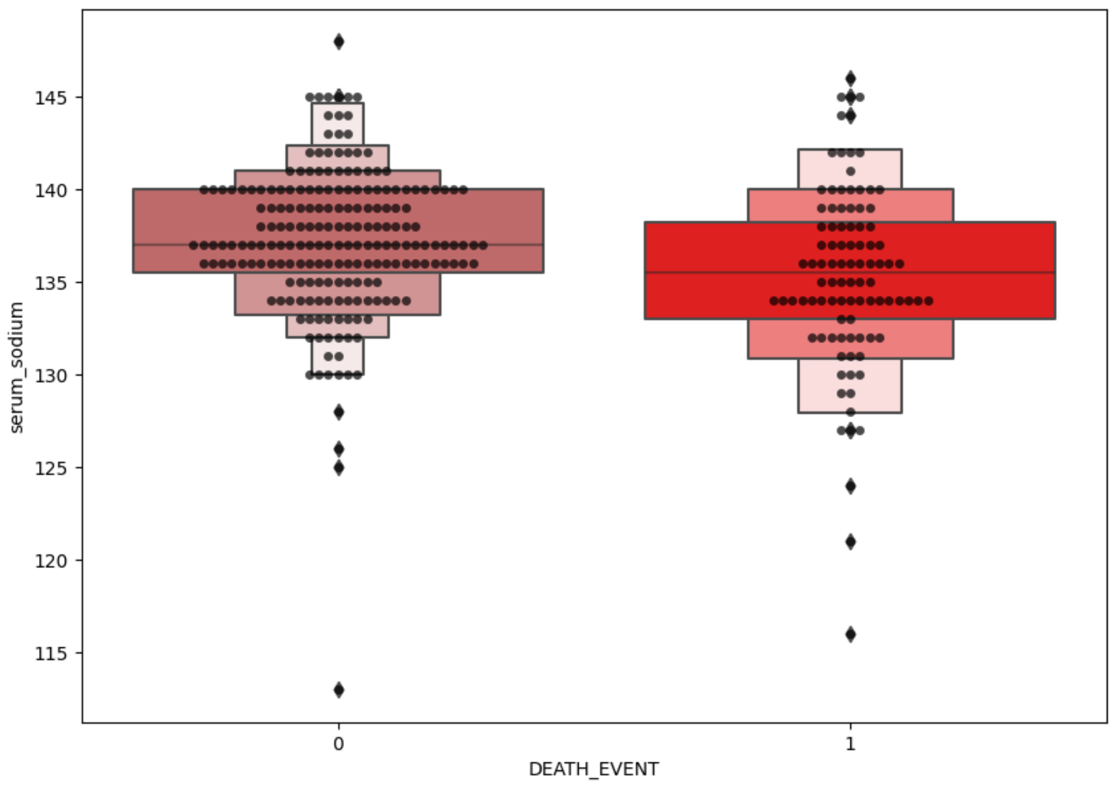

80
70
60
50
40
30
20
ejection_fraction
0
1
DEATH_EVENT

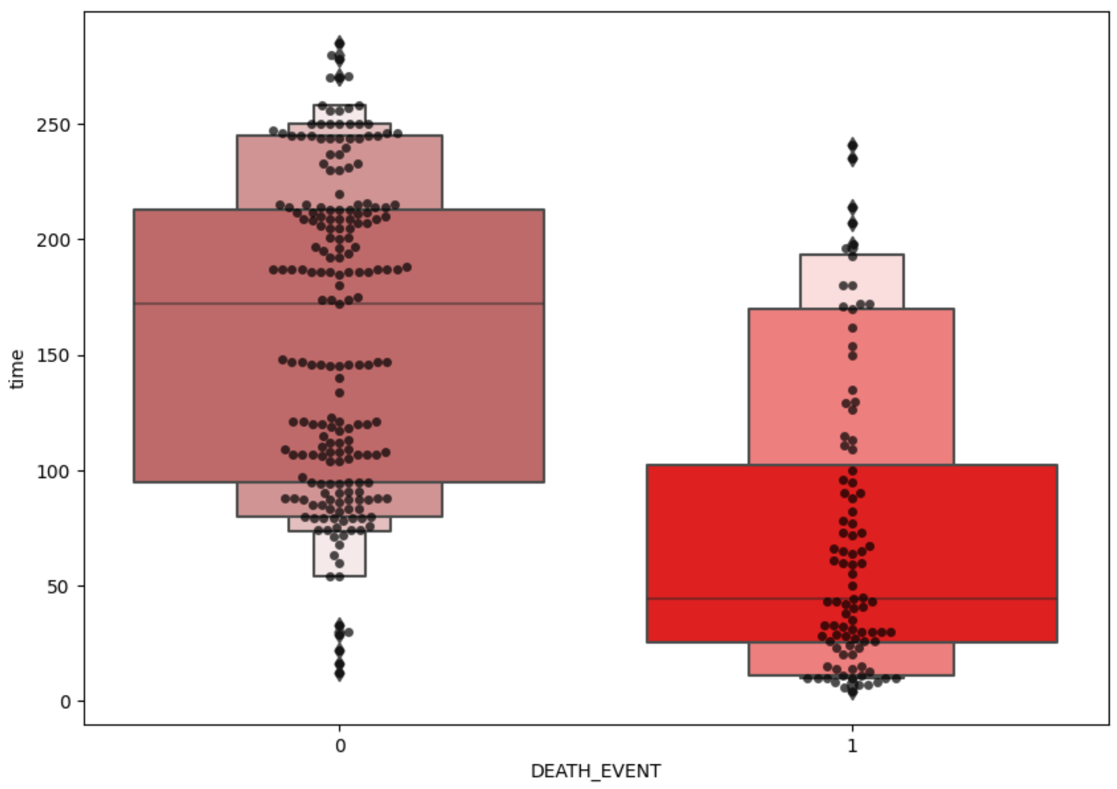


800000
600000
400000
200000
0
platelets
0
1
DEATH_EVENT



8
6
4
2
serum_creatinine
0
1
DEATH_EVENT


145
140
135
130
125
120
115
serum_sodium
0
1
DEATH_EVENT

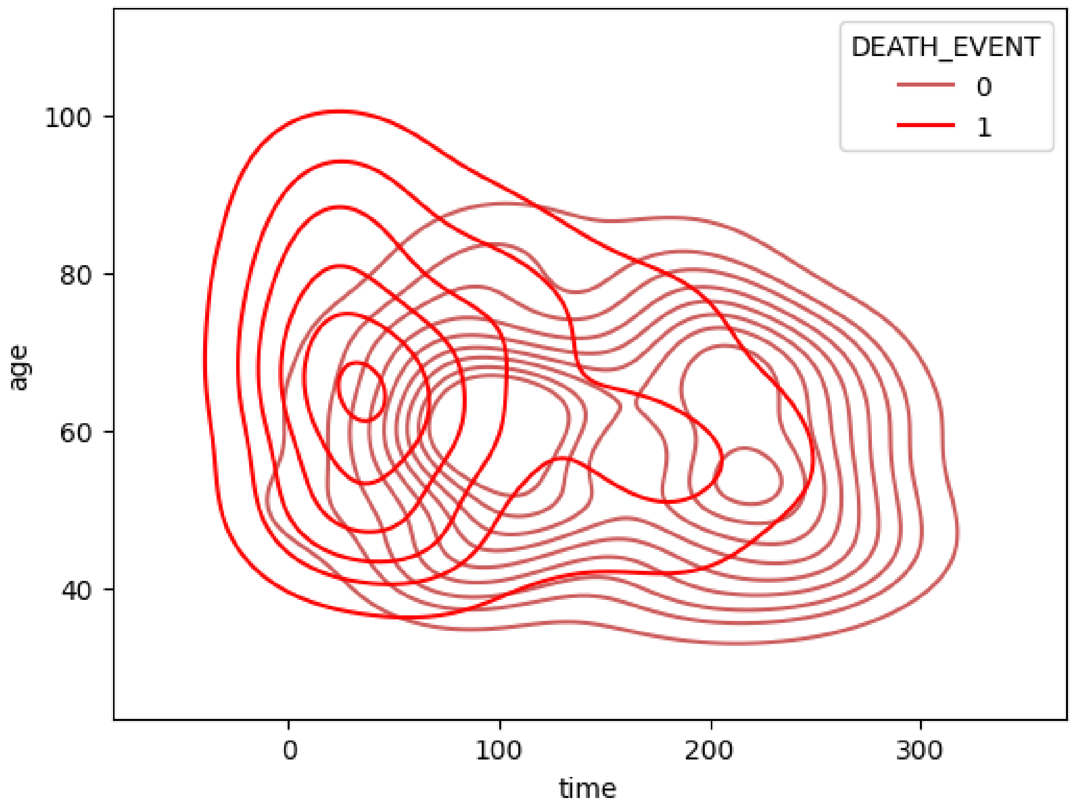


0.0.13 Note:

- Few Outliers can be seen in almost all the features
- Considering the size of the dataset and relevancy of it, we won't be dropping such outliers in data preprocessing which wouldn't bring any statistical fluke.

[11]:
```
# Plotting "Kernel Density Estimation (kde plot)" of time and age features - ␣
↪both of which are significant ones.
sns.kdeplot(x=data_df["time"], y=data_df["age"], hue =data_df["DEATH_EVENT"],␣
↪palette=cols)
```

[11]:
```
<Axes: xlabel='time', ylabel='age'>
```



0.0.14 Note:

- With less follow-up days, patients often died only when they aged more.
- More the follow-up days more the probability is, of any fatality.

DATA PREPROCESSING

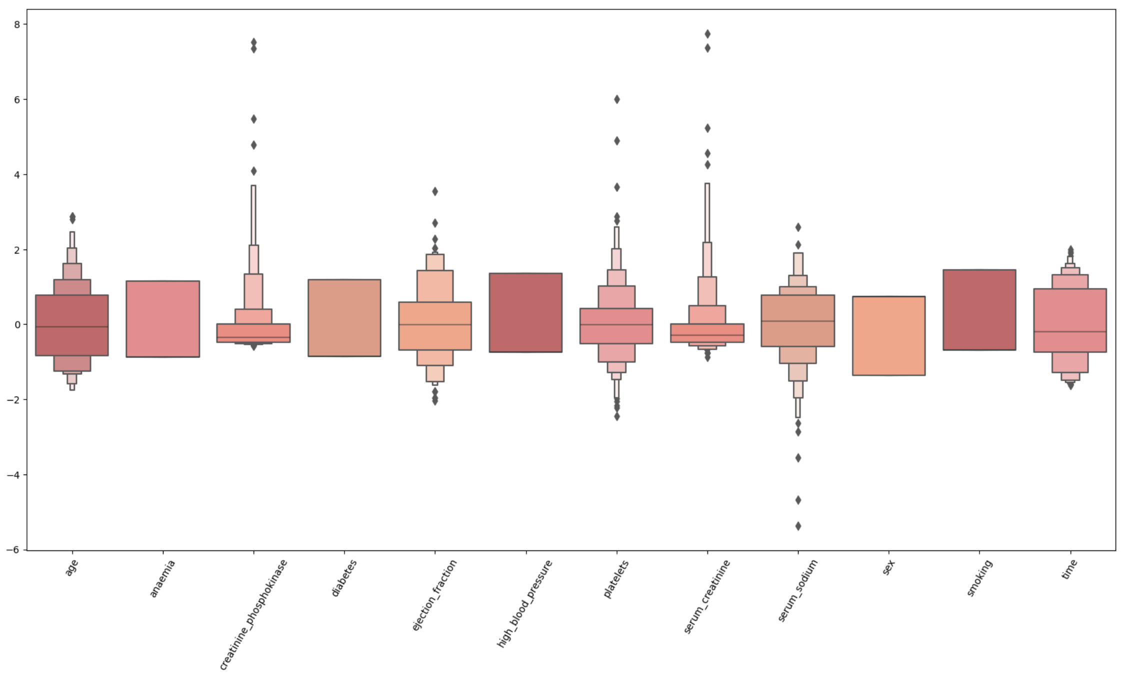
```
[12]: # Defining independent and dependent attributes in training and test sets
      X=data_df.drop(["DEATH_EVENT"],axis=1)
      y=data_df["DEATH_EVENT"]
```

```
[13]: # Setting up a standard scaler for the features and analyzing it thereafter
      col_names = list(X.columns)
      s_scaler = preprocessing.StandardScaler()
      X_scaled= s_scaler.fit_transform(X)
      X_scaled = pd.DataFrame(X_scaled, columns=col_names)
      X_scaled.describe().T
```

```
[13]:                           count          mean       std       min       25%  \
      age                       299.0  5.703353e-16  1.001676 -1.754448 -0.828124
      anaemia                   299.0  1.009969e-16  1.001676 -0.871105 -0.871105
      creatinine_phosphokinase  299.0  0.000000e+00  1.001676 -0.576918 -0.480393
```

```
diabetes                  299.0  9.060014e-17  1.001676 -0.847579 -0.847579
ejection_fraction         299.0 -3.267546e-17  1.001676 -2.038387 -0.684180
high_blood_pressure       299.0  0.000000e+00  1.001676 -0.735688 -0.735688
platelets                 299.0  7.723291e-17  1.001676 -2.440155 -0.520870
serum_creatinine          299.0  1.425838e-16  1.001676 -0.865509 -0.478205
serum_sodium              299.0 -8.673849e-16  1.001676 -5.363206 -0.595996
sex                       299.0 -8.911489e-18  1.001676 -1.359272 -1.359272
smoking                   299.0 -1.188199e-17  1.001676 -0.687682 -0.687682
time                      299.0 -1.901118e-16  1.001676 -1.629502 -0.739000

                               50%       75%       max
age                      -0.070223  0.771889  2.877170
anaemia                  -0.871105  1.147968  1.147968
creatinine_phosphokinase -0.342574  0.000166  7.514640
diabetes                 -0.847579  1.179830  1.179830
ejection_fraction        -0.007077  0.585389  3.547716
high_blood_pressure      -0.735688  1.359272  1.359272
platelets                -0.013908  0.411120  6.008180
serum_creatinine         -0.284552  0.005926  7.752020
serum_sodium              0.085034  0.766064  2.582144
sex                       0.735688  0.735688  0.735688
smoking                  -0.687682  1.454161  1.454161
time                     -0.196954  0.938759  1.997038
```

[14]:
```
#Plotting the scaled features using boxen plots
colors =["#CD5C5C","#F08080","#FA8072","#E9967A","#FFA07A"]
plt.figure(figsize=(20,10))
sns.boxenplot(data = X_scaled,palette = colors)
plt.xticks(rotation=60)
plt.show()
```



[15]:
```
#spliting variables into training and test sets
X_train, X_test, y_train,y_test = train_test_split(X_scaled,y,test_size=0.
↪30,random_state=25)
```

MODEL BUILDING

0.1 1. SUPPORT VECTOR MACHINE (SVM)

[16]:
```
# Instantiating the SVM algorithm
model1=svm.SVC()

# Fitting the model
model1.fit (X_train, y_train)

# Predicting the test variables
y_pred = model1.predict(X_test)

# Getting the score
model1.score (X_test, y_test)
```

[16]:
```
0.7888888888888889
```

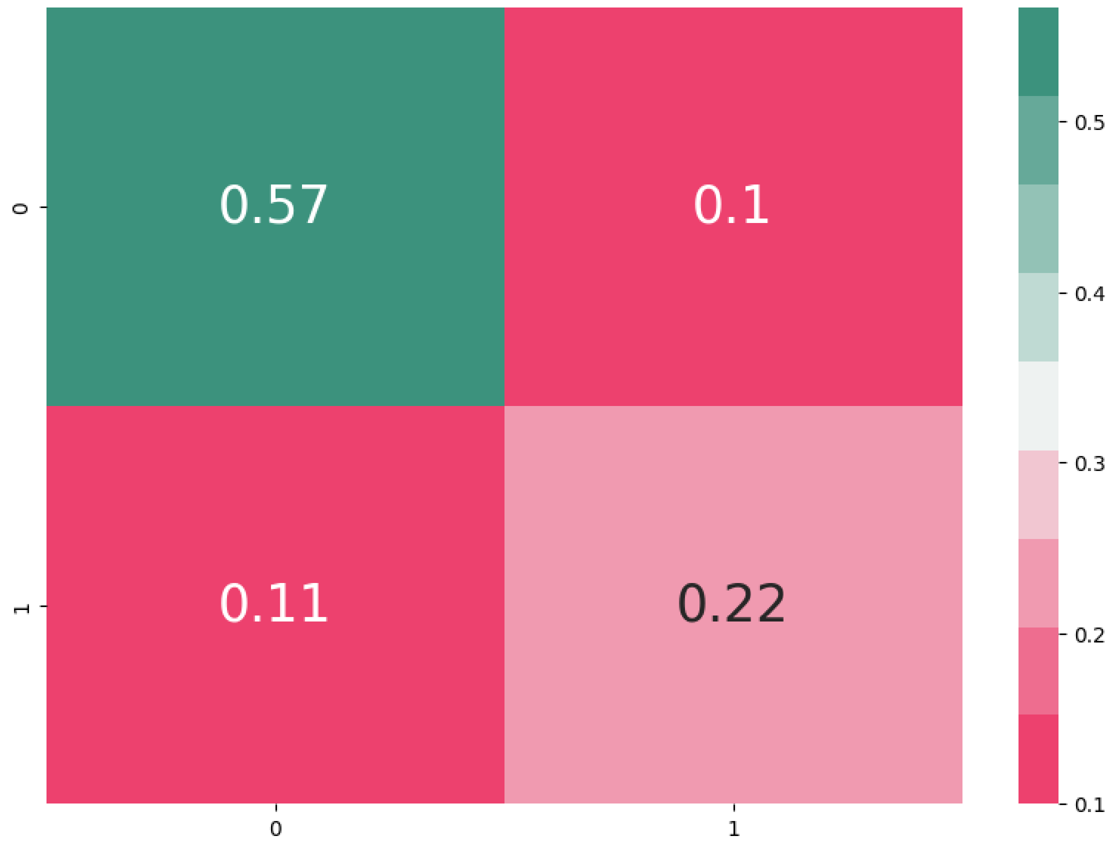
[17]:
```
# Printing classification report (since there was biasness in target labels)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support
```

```
           0       0.84      0.85      0.84        60
           1       0.69      0.67      0.68        30

    accuracy                           0.79        90
   macro avg       0.76      0.76      0.76        90
weighted avg       0.79      0.79      0.79        90
```

[18]:
```
# Getting the confusion matrix
cmap1 = sns.diverging_palette(2, 165, s=80, l=55, n=9)
plt.subplots(figsize=(10,7))
cf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(cf_matrix/np.sum(cf_matrix), cmap = cmap1, annot = True, annot_kws␣
 ↪= {'size':25})
```

[18]:
```
<Axes: >
```



0.2 2. Artificial Neural Network (ANN)

[22]:
```
early_stopping = callbacks.EarlyStopping(
    min_delta=0.001, # minimium amount of change to count as an improvement
    patience=20, # how many epochs to wait before stopping
    restore_best_weights=True)

# Initialising the NN
model = Sequential()

# layers
model.add(Dense(units = 16, kernel_initializer = 'uniform', activation =␣
 ↪'relu', input_dim = 12))
model.add(Dense(units = 8, kernel_initializer = 'uniform', activation = 'relu'))
model.add(Dropout(0.25))
model.add(Dense(units = 8, kernel_initializer = 'uniform', activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(units = 1, kernel_initializer = 'uniform', activation =␣
 ↪'sigmoid'))

# Compiling the ANN
model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics =␣
 ↪['accuracy'])

# Train the ANN
history = model.fit(X_train, y_train, batch_size = 25, epochs =␣
 ↪80,callbacks=[early_stopping], validation_split=0.25)
```

```
Epoch 1/80
7/7 [==============================] - 2s 61ms/step - loss: 0.6928 - accuracy:
0.6346 - val_loss: 0.6906 - val_accuracy: 0.8302
Epoch 2/80
7/7 [==============================] - 0s 15ms/step - loss: 0.6918 - accuracy:
0.6346 - val_loss: 0.6884 - val_accuracy: 0.8302
Epoch 3/80
7/7 [==============================] - 0s 11ms/step - loss: 0.6907 - accuracy:
0.6346 - val_loss: 0.6859 - val_accuracy: 0.8302
Epoch 4/80
7/7 [==============================] - 0s 15ms/step - loss: 0.6897 - accuracy:
0.6346 - val_loss: 0.6832 - val_accuracy: 0.8302
Epoch 5/80
7/7 [==============================] - 0s 15ms/step - loss: 0.6888 - accuracy:
0.6346 - val_loss: 0.6800 - val_accuracy: 0.8302
Epoch 6/80
7/7 [==============================] - 0s 14ms/step - loss: 0.6870 - accuracy:
0.6346 - val_loss: 0.6766 - val_accuracy: 0.8302
Epoch 7/80
```

```
7/7 [==============================] - 0s 20ms/step - loss: 0.6851 - accuracy:
0.6346 - val_loss: 0.6724 - val_accuracy: 0.8302
Epoch 8/80
7/7 [==============================] - 0s 11ms/step - loss: 0.6824 - accuracy:
0.6346 - val_loss: 0.6665 - val_accuracy: 0.8302
Epoch 9/80
7/7 [==============================] - 0s 14ms/step - loss: 0.6778 - accuracy:
0.6346 - val_loss: 0.6590 - val_accuracy: 0.8302
Epoch 10/80
7/7 [==============================] - 0s 13ms/step - loss: 0.6750 - accuracy:
0.6346 - val_loss: 0.6489 - val_accuracy: 0.8302
Epoch 11/80
7/7 [==============================] - 0s 13ms/step - loss: 0.6672 - accuracy:
0.6346 - val_loss: 0.6355 - val_accuracy: 0.8302
Epoch 12/80
7/7 [==============================] - 0s 13ms/step - loss: 0.6573 - accuracy:
0.6346 - val_loss: 0.6176 - val_accuracy: 0.8302
Epoch 13/80
7/7 [==============================] - 0s 12ms/step - loss: 0.6462 - accuracy:
0.6346 - val_loss: 0.5961 - val_accuracy: 0.8302
Epoch 14/80
7/7 [==============================] - 0s 14ms/step - loss: 0.6322 - accuracy:
0.6346 - val_loss: 0.5722 - val_accuracy: 0.8302
Epoch 15/80
7/7 [==============================] - 0s 12ms/step - loss: 0.6098 - accuracy:
0.6346 - val_loss: 0.5430 - val_accuracy: 0.8302
Epoch 16/80
7/7 [==============================] - 0s 11ms/step - loss: 0.5960 - accuracy:
0.6346 - val_loss: 0.5134 - val_accuracy: 0.8302
Epoch 17/80
7/7 [==============================] - 0s 13ms/step - loss: 0.5802 - accuracy:
0.6346 - val_loss: 0.4791 - val_accuracy: 0.8302
Epoch 18/80
7/7 [==============================] - 0s 26ms/step - loss: 0.5496 - accuracy:
0.6346 - val_loss: 0.4454 - val_accuracy: 0.8302
Epoch 19/80
7/7 [==============================] - 0s 14ms/step - loss: 0.5249 - accuracy:
0.6346 - val_loss: 0.4151 - val_accuracy: 0.8302
Epoch 20/80
7/7 [==============================] - 0s 16ms/step - loss: 0.5170 - accuracy:
0.6346 - val_loss: 0.3859 - val_accuracy: 0.8302
Epoch 21/80
7/7 [==============================] - 0s 16ms/step - loss: 0.4942 - accuracy:
0.6346 - val_loss: 0.3596 - val_accuracy: 0.8302
Epoch 22/80
7/7 [==============================] - 0s 11ms/step - loss: 0.4802 - accuracy:
0.6346 - val_loss: 0.3414 - val_accuracy: 0.8302
Epoch 23/80
```

```
7/7 [==============================] - 0s 14ms/step - loss: 0.5169 - accuracy:
0.6346 - val_loss: 0.3279 - val_accuracy: 0.8302
Epoch 24/80
7/7 [==============================] - 0s 12ms/step - loss: 0.4919 - accuracy:
0.6346 - val_loss: 0.3237 - val_accuracy: 0.8302
Epoch 25/80
7/7 [==============================] - 0s 14ms/step - loss: 0.5012 - accuracy:
0.6346 - val_loss: 0.3206 - val_accuracy: 0.8302
Epoch 26/80
7/7 [==============================] - 0s 13ms/step - loss: 0.4810 - accuracy:
0.6346 - val_loss: 0.3183 - val_accuracy: 0.8302
Epoch 27/80
7/7 [==============================] - 0s 20ms/step - loss: 0.4781 - accuracy:
0.6346 - val_loss: 0.3131 - val_accuracy: 0.8302
Epoch 28/80
7/7 [==============================] - 0s 14ms/step - loss: 0.4871 - accuracy:
0.6346 - val_loss: 0.3109 - val_accuracy: 0.8302
Epoch 29/80
7/7 [==============================] - 0s 14ms/step - loss: 0.4921 - accuracy:
0.6346 - val_loss: 0.3081 - val_accuracy: 0.8302
Epoch 30/80
7/7 [==============================] - 0s 20ms/step - loss: 0.4854 - accuracy:
0.6346 - val_loss: 0.3097 - val_accuracy: 0.8302
Epoch 31/80
7/7 [==============================] - 0s 31ms/step - loss: 0.4851 - accuracy:
0.6346 - val_loss: 0.3090 - val_accuracy: 0.8302
Epoch 32/80
7/7 [==============================] - 0s 27ms/step - loss: 0.4815 - accuracy:
0.6346 - val_loss: 0.3068 - val_accuracy: 0.8302
Epoch 33/80
7/7 [==============================] - 0s 28ms/step - loss: 0.4646 - accuracy:
0.6346 - val_loss: 0.3031 - val_accuracy: 0.8302
Epoch 34/80
7/7 [==============================] - 0s 27ms/step - loss: 0.4735 - accuracy:
0.6346 - val_loss: 0.2996 - val_accuracy: 0.8302
Epoch 35/80
7/7 [==============================] - 0s 7ms/step - loss: 0.4759 - accuracy:
0.6346 - val_loss: 0.2945 - val_accuracy: 0.8302
Epoch 36/80
7/7 [==============================] - 0s 10ms/step - loss: 0.4567 - accuracy:
0.6346 - val_loss: 0.2940 - val_accuracy: 0.8302
Epoch 37/80
7/7 [==============================] - 0s 7ms/step - loss: 0.4543 - accuracy:
0.6346 - val_loss: 0.2942 - val_accuracy: 0.8302
Epoch 38/80
7/7 [==============================] - 0s 8ms/step - loss: 0.4656 - accuracy:
0.6346 - val_loss: 0.2912 - val_accuracy: 0.8302
Epoch 39/80
```

```
7/7 [==============================] - 0s 10ms/step - loss: 0.4544 - accuracy:
0.6346 - val_loss: 0.2899 - val_accuracy: 0.8302
Epoch 40/80
7/7 [==============================] - 0s 7ms/step - loss: 0.4734 - accuracy:
0.6346 - val_loss: 0.2884 - val_accuracy: 0.8302
Epoch 41/80
7/7 [==============================] - 0s 10ms/step - loss: 0.4450 - accuracy:
0.6603 - val_loss: 0.2851 - val_accuracy: 0.8679
Epoch 42/80
7/7 [==============================] - 0s 7ms/step - loss: 0.4495 - accuracy:
0.7756 - val_loss: 0.2835 - val_accuracy: 0.8679
Epoch 43/80
7/7 [==============================] - 0s 10ms/step - loss: 0.4590 - accuracy:
0.7628 - val_loss: 0.2831 - val_accuracy: 0.8679
Epoch 44/80
7/7 [==============================] - 0s 10ms/step - loss: 0.4551 - accuracy:
0.7628 - val_loss: 0.2833 - val_accuracy: 0.8679
Epoch 45/80
7/7 [==============================] - 0s 8ms/step - loss: 0.4672 - accuracy:
0.7821 - val_loss: 0.2824 - val_accuracy: 0.8679
Epoch 46/80
7/7 [==============================] - 0s 10ms/step - loss: 0.4471 - accuracy:
0.7885 - val_loss: 0.2835 - val_accuracy: 0.8679
Epoch 47/80
7/7 [==============================] - 0s 8ms/step - loss: 0.4438 - accuracy:
0.7756 - val_loss: 0.2849 - val_accuracy: 0.8868
Epoch 48/80
7/7 [==============================] - 0s 9ms/step - loss: 0.4608 - accuracy:
0.8333 - val_loss: 0.2852 - val_accuracy: 0.8868
Epoch 49/80
7/7 [==============================] - 0s 10ms/step - loss: 0.4173 - accuracy:
0.8462 - val_loss: 0.2836 - val_accuracy: 0.8868
Epoch 50/80
7/7 [==============================] - 0s 10ms/step - loss: 0.4323 - accuracy:
0.8013 - val_loss: 0.2813 - val_accuracy: 0.8868
Epoch 51/80
7/7 [==============================] - 0s 10ms/step - loss: 0.4268 - accuracy:
0.8013 - val_loss: 0.2810 - val_accuracy: 0.8868
Epoch 52/80
7/7 [==============================] - 0s 9ms/step - loss: 0.4202 - accuracy:
0.8462 - val_loss: 0.2812 - val_accuracy: 0.8868
Epoch 53/80
7/7 [==============================] - 0s 8ms/step - loss: 0.4390 - accuracy:
0.8397 - val_loss: 0.2797 - val_accuracy: 0.8868
Epoch 54/80
7/7 [==============================] - 0s 9ms/step - loss: 0.4262 - accuracy:
0.8462 - val_loss: 0.2791 - val_accuracy: 0.8868
Epoch 55/80
```

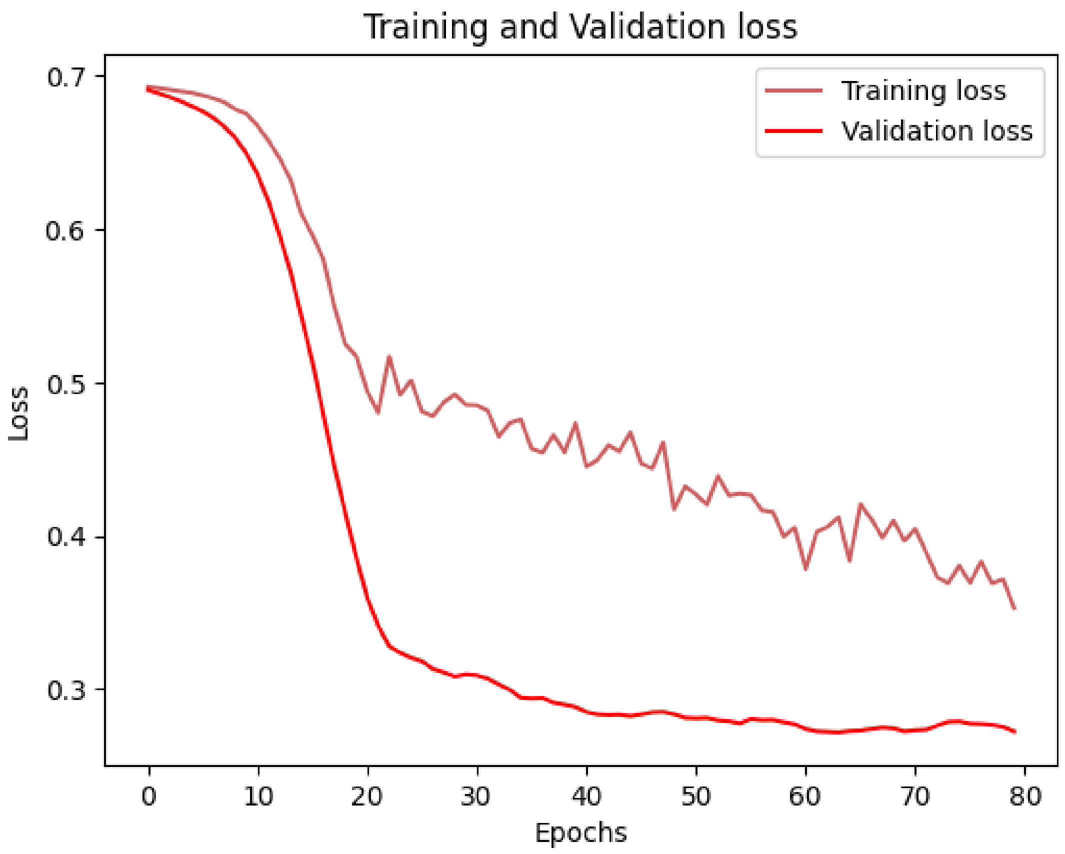
```
7/7 [==============================] - 0s 10ms/step - loss: 0.4274 - accuracy:
0.8462 - val_loss: 0.2777 - val_accuracy: 0.8868
Epoch 56/80
7/7 [==============================] - 0s 7ms/step - loss: 0.4265 - accuracy:
0.8526 - val_loss: 0.2806 - val_accuracy: 0.8679
Epoch 57/80
7/7 [==============================] - 0s 7ms/step - loss: 0.4166 - accuracy:
0.8590 - val_loss: 0.2798 - val_accuracy: 0.8679
Epoch 58/80
7/7 [==============================] - 0s 7ms/step - loss: 0.4155 - accuracy:
0.8590 - val_loss: 0.2800 - val_accuracy: 0.8679
Epoch 59/80
7/7 [==============================] - 0s 10ms/step - loss: 0.3995 - accuracy:
0.8654 - val_loss: 0.2785 - val_accuracy: 0.8679
Epoch 60/80
7/7 [==============================] - 0s 7ms/step - loss: 0.4054 - accuracy:
0.8526 - val_loss: 0.2771 - val_accuracy: 0.8679
Epoch 61/80
7/7 [==============================] - 0s 10ms/step - loss: 0.3782 - accuracy:
0.8654 - val_loss: 0.2740 - val_accuracy: 0.8679
Epoch 62/80
7/7 [==============================] - 0s 10ms/step - loss: 0.4024 - accuracy:
0.8782 - val_loss: 0.2726 - val_accuracy: 0.8679
Epoch 63/80
7/7 [==============================] - 0s 9ms/step - loss: 0.4059 - accuracy:
0.8526 - val_loss: 0.2722 - val_accuracy: 0.8679
Epoch 64/80
7/7 [==============================] - 0s 7ms/step - loss: 0.4120 - accuracy:
0.8397 - val_loss: 0.2719 - val_accuracy: 0.8679
Epoch 65/80
7/7 [==============================] - 0s 7ms/step - loss: 0.3837 - accuracy:
0.8590 - val_loss: 0.2727 - val_accuracy: 0.8679
Epoch 66/80
7/7 [==============================] - 0s 10ms/step - loss: 0.4206 - accuracy:
0.8590 - val_loss: 0.2731 - val_accuracy: 0.8679
Epoch 67/80
7/7 [==============================] - 0s 10ms/step - loss: 0.4106 - accuracy:
0.8590 - val_loss: 0.2741 - val_accuracy: 0.8679
Epoch 68/80
7/7 [==============================] - 0s 10ms/step - loss: 0.3988 - accuracy:
0.8526 - val_loss: 0.2751 - val_accuracy: 0.8679
Epoch 69/80
7/7 [==============================] - 0s 10ms/step - loss: 0.4098 - accuracy:
0.8846 - val_loss: 0.2745 - val_accuracy: 0.8679
Epoch 70/80
7/7 [==============================] - 0s 7ms/step - loss: 0.3968 - accuracy:
0.8846 - val_loss: 0.2725 - val_accuracy: 0.8679
Epoch 71/80
```

```
7/7 [==============================] - 0s 8ms/step - loss: 0.4044 - accuracy:
0.8654 - val_loss: 0.2733 - val_accuracy: 0.8679
Epoch 72/80
7/7 [==============================] - 0s 7ms/step - loss: 0.3889 - accuracy:
0.8654 - val_loss: 0.2736 - val_accuracy: 0.8679
Epoch 73/80
7/7 [==============================] - 0s 9ms/step - loss: 0.3730 - accuracy:
0.8590 - val_loss: 0.2763 - val_accuracy: 0.8679
Epoch 74/80
7/7 [==============================] - 0s 7ms/step - loss: 0.3690 - accuracy:
0.8654 - val_loss: 0.2787 - val_accuracy: 0.8491
Epoch 75/80
7/7 [==============================] - 0s 8ms/step - loss: 0.3805 - accuracy:
0.8590 - val_loss: 0.2790 - val_accuracy: 0.8679
Epoch 76/80
7/7 [==============================] - 0s 7ms/step - loss: 0.3693 - accuracy:
0.8782 - val_loss: 0.2775 - val_accuracy: 0.8679
Epoch 77/80
7/7 [==============================] - 0s 7ms/step - loss: 0.3832 - accuracy:
0.8654 - val_loss: 0.2773 - val_accuracy: 0.8679
Epoch 78/80
7/7 [==============================] - 0s 8ms/step - loss: 0.3690 - accuracy:
0.8846 - val_loss: 0.2767 - val_accuracy: 0.8679
Epoch 79/80
7/7 [==============================] - 0s 7ms/step - loss: 0.3716 - accuracy:
0.8974 - val_loss: 0.2754 - val_accuracy: 0.8679
Epoch 80/80
7/7 [==============================] - 0s 7ms/step - loss: 0.3530 - accuracy:
0.8846 - val_loss: 0.2725 - val_accuracy: 0.8679
```

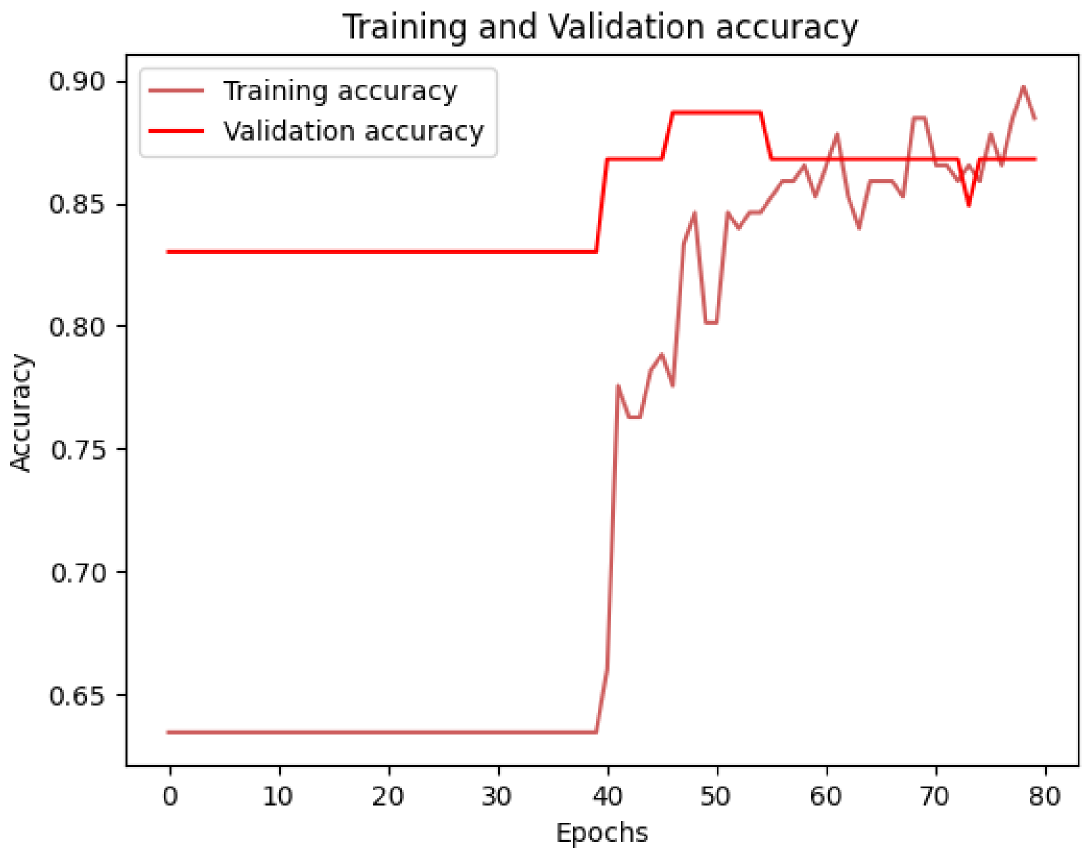
[23]:
```
val_accuracy = np.mean(history.history['val_accuracy'])
print("\n%s: %.2f%%" % ('val_accuracy is', val_accuracy*100))
```

```
val_accuracy is: 85.09%
```

[24]:
```
history_df = pd.DataFrame(history.history)

plt.plot(history_df.loc[:, ['loss']], "#CD5C5C", label='Training loss')
plt.plot(history_df.loc[:, ['val_loss']],"#FF0000", label='Validation loss')
plt.title('Training and Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(loc="best")

plt.show()
```



[25]:
```
history_df = pd.DataFrame(history.history)

plt.plot(history_df.loc[:, ['accuracy']], "#CD5C5C", label='Training accuracy')
plt.plot(history_df.loc[:, ['val_accuracy']],"#FF0000", label='Validation␣
↪accuracy')

plt.title('Training and Validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

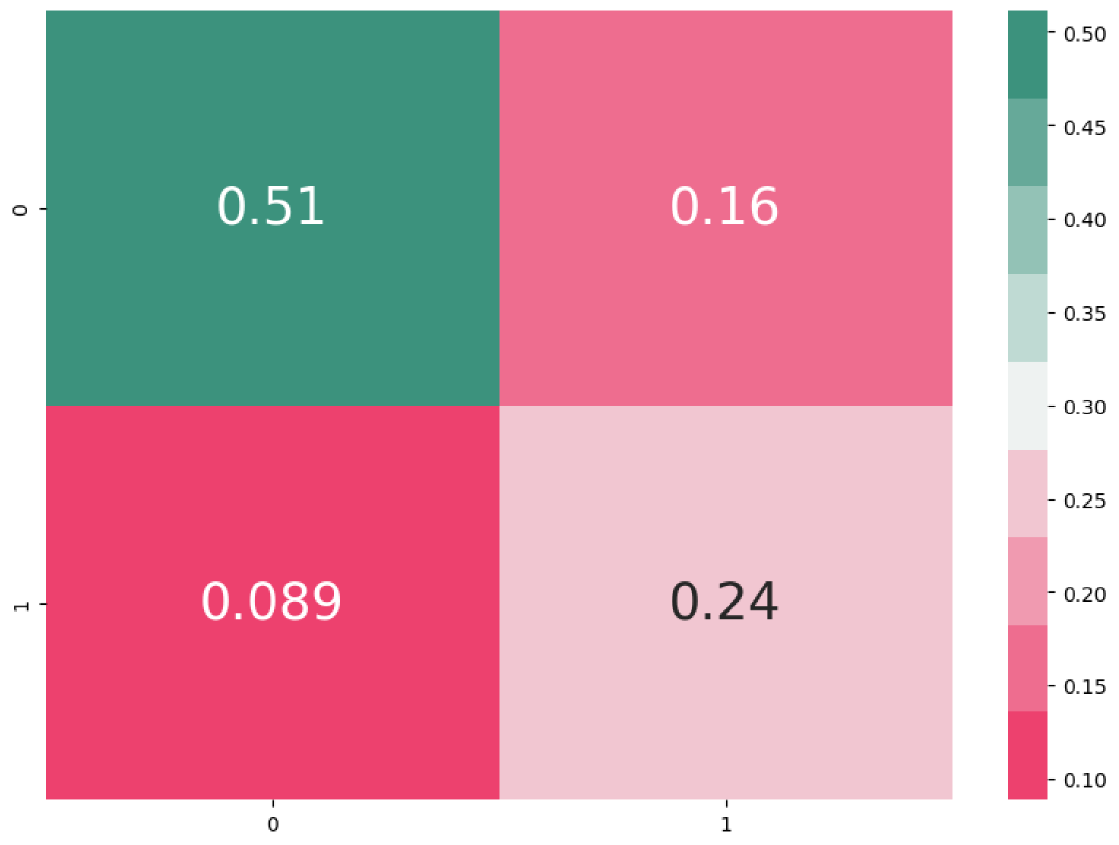


[26]:
```
# Predicting the test set results
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.4)
np.set_printoptions()
```

```
3/3 [==============================] - 0s 3ms/step
```

[27]:
```
# Getting the confusion matrix
cmap1 = sns.diverging_palette(2, 165, s=80, l=55, n=9)
plt.subplots(figsize=(10,7))
cf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(cf_matrix/np.sum(cf_matrix), cmap = cmap1, annot = True, annot_kws␣
 ↪= {'size':25})
```

[27]:
```
<Axes: >
```



[29]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.77      0.81        60
           1       0.61      0.73      0.67        30

    accuracy                           0.76        90
   macro avg       0.73      0.75      0.74        90
weighted avg       0.77      0.76      0.76        90
```

[ ]: