In [ ]:
```
# !pip list
```

In [ ]:
```
# Step 1 : Importing libraries
```

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

In [ ]:
```
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

In [ ]:
```
from sklearn import svm
from sklearn.metrics import accuracy_score,recall_score,confusion_matrix,f1_sc
from tensorflow.keras.layers import Dense,BatchNormalization,Dropout, LSTM
from tensorflow.keras.models import Sequential
from tensorflow.keras import callbacks
```

In [ ]:
```
# Step 2 : Data importing and analysis
```

In [ ]:
```
from google.colab import files
import pandas as pd
import io

# Upload file
uploaded = files.upload()

# Check the keys (filenames)
print("Uploaded file(s):", uploaded.keys())

# Use the actual file name; if there's only one, use the first key.
filename = list(uploaded.keys())[0]
df = pd.read_csv(io.BytesIO(uploaded[filename]))

# Display the first few rows of the DataFrame
df.head()
```

Browse... Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving heart rate detection.csv to heart rate detection (3).csv
Uploaded file(s): dict_keys(['heart rate detection (3).csv'])
```

Out[ ]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_bl
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	

In [ ]:

```
data_df = pd.read_csv('heart rate detection.csv')
```

In [ ]:

```
data_df
```

Out[ ]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_
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	
...	...	...	...	...	...	
294	62.0	0	61	1	38	
295	55.0	0	1820	0	38	
296	45.0	0	2060	1	60	
297	45.0	0	2413	0	38	
298	50.0	0	196	0	45	

299 rows × 13 columns

In [ ]:

```
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
```

In [ ]:
```
data_df.isnull().sum()
```

Out[ ]:

	0
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

In [ ]:
```
print(data_df.columns)
```

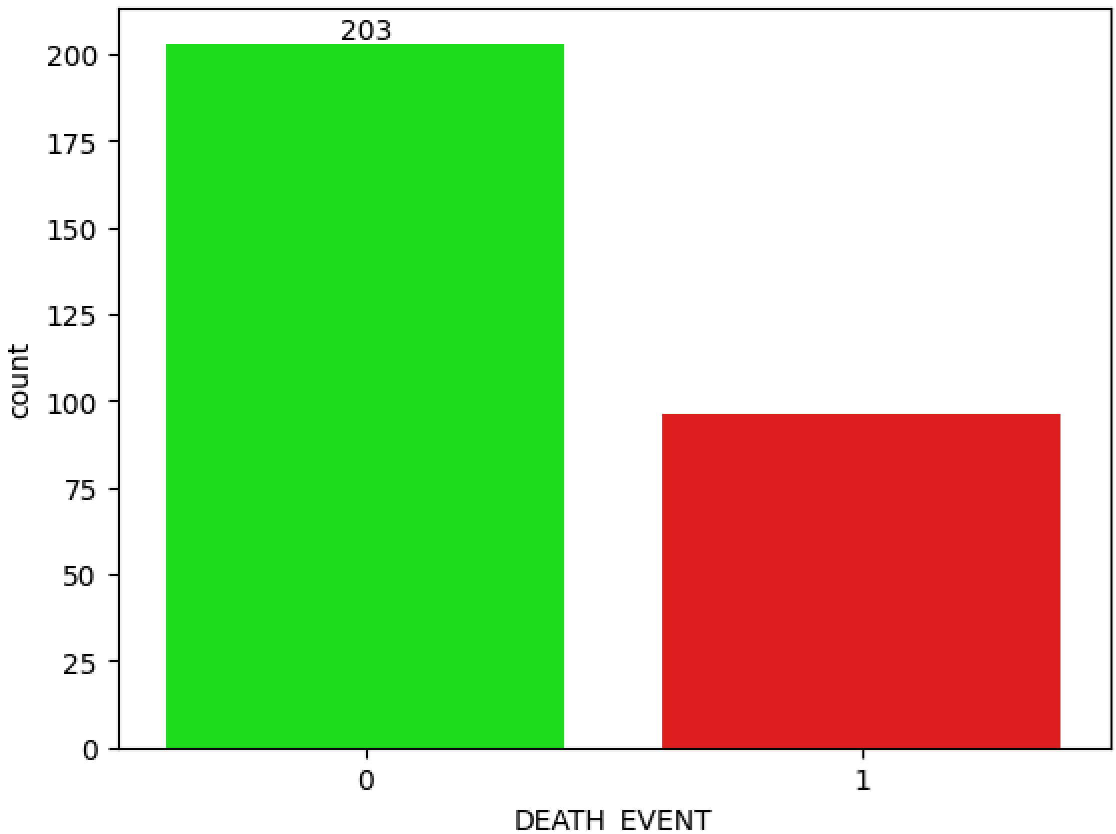
```
Index(['age', 'anaemia', 'creatinine_phosphokinase', 'diabetes',
       'ejection_fraction', 'high_blood_pressure', 'platelets',
       'serum_creatinine', 'serum_sodium', 'sex', 'smoking', 'time',
       'DEATH_EVENT'],
      dtype='object')
```

In [ ]:

```
cols = ["#00FF00","#FF0000"]
ax = sns.countplot(x=data_df["DEATH_EVENT"],palette=cols)
ax.bar_label(ax.containers[0])
```

```
<ipython-input-270-d98c30379995>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same e
ffect.

  ax = sns.countplot(x=data_df["DEATH_EVENT"],palette=cols)
```

Out[ ]: [Text(0, 0, '203')]



In [ ]:

```
data_df.describe().T
```

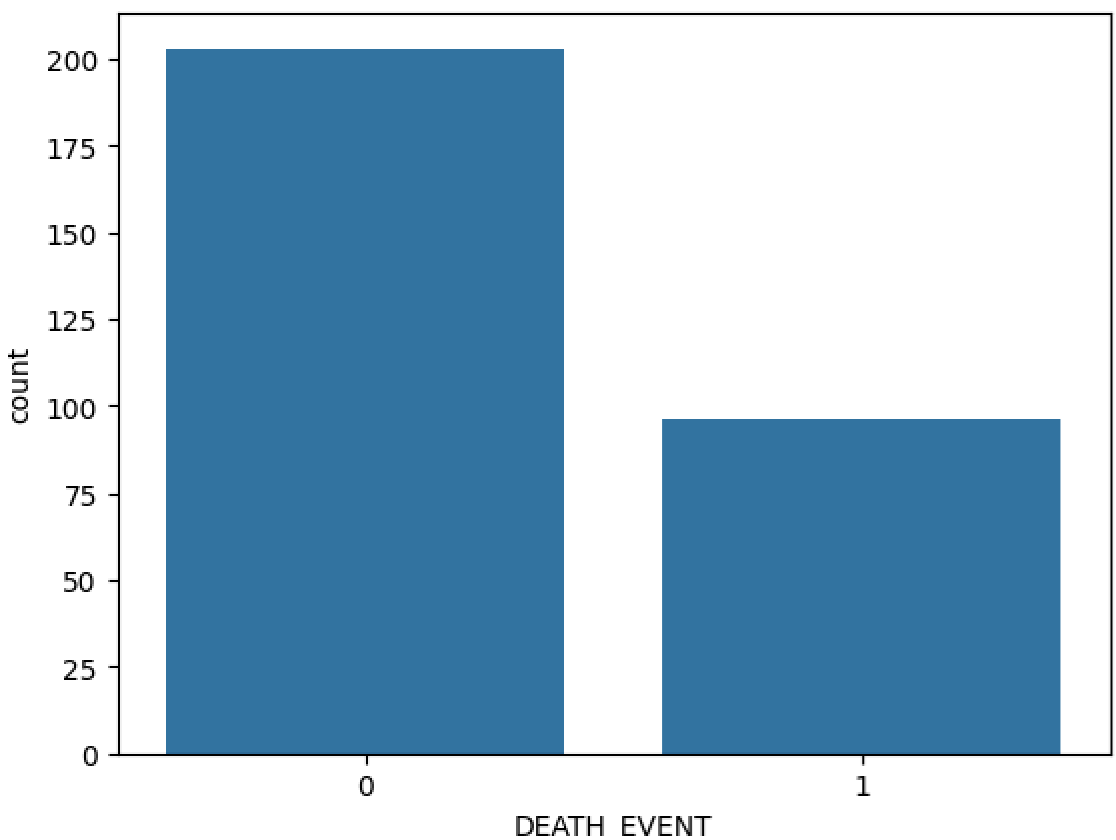
Out[ ]:

	count	mean	std	min	25%
age	299.0	60.833893	11.894809	40.0	51.0
anaemia	299.0	0.431438	0.496107	0.0	0.0
creatinine_phosphokinase	299.0	581.839465	970.287881	23.0	116.5
diabetes	299.0	0.418060	0.494067	0.0	0.0
ejection_fraction	299.0	38.083612	11.834841	14.0	30.0
high_blood_pressure	299.0	0.351171	0.478136	0.0	0.0
platelets	299.0	263358.029264	97804.236869	25100.0	212500.0
serum_creatinine	299.0	1.393880	1.034510	0.5	0.9
serum_sodium	299.0	136.625418	4.412477	113.0	134.0
sex	299.0	0.648829	0.478136	0.0	0.0
smoking	299.0	0.321070	0.467670	0.0	0.0
time	299.0	130.260870	77.614208	4.0	73.0
DEATH_EVENT	299.0	0.321070	0.467670	0.0	0.0

In [ ]:

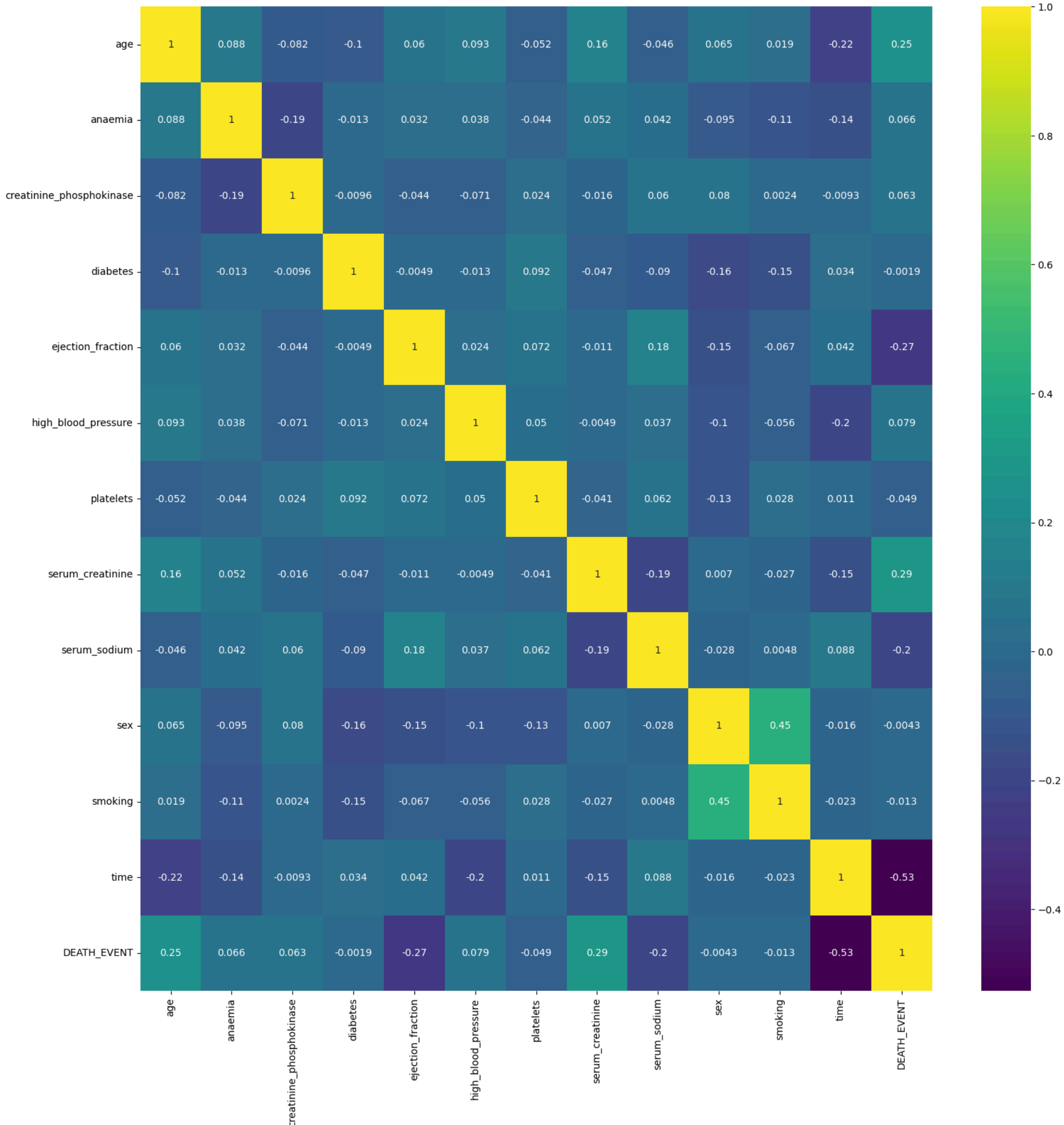
```
sns.countplot(x=data_df['DEATH_EVENT'])
```

Out[ ]: <Axes: xlabel='DEATH_EVENT', ylabel='count'>



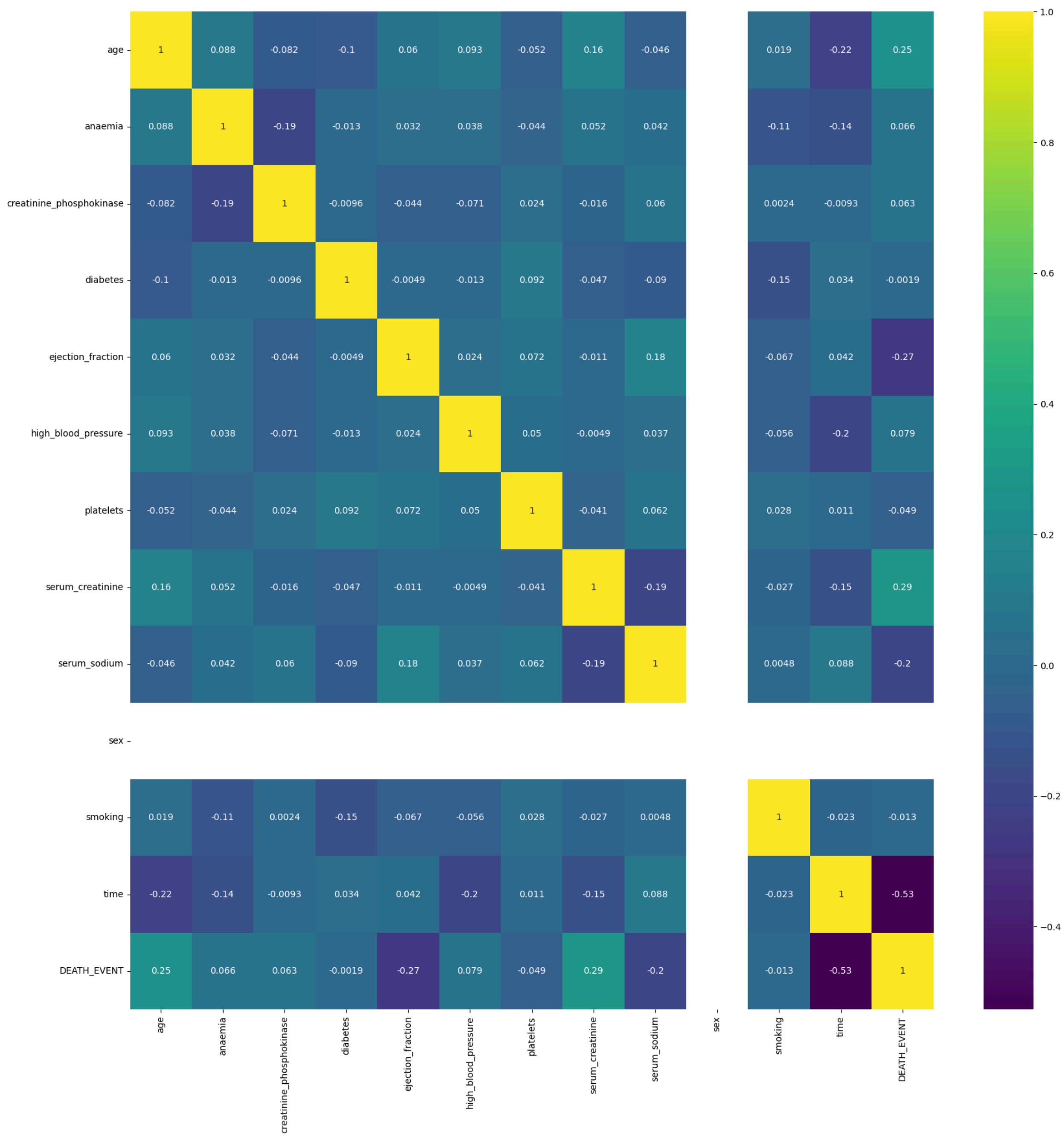
In [ ]:

```
plt.figure(figsize=(18,18))
sns.heatmap(data_df.corr(),annot=True,cmap='viridis')
plt.show()
```



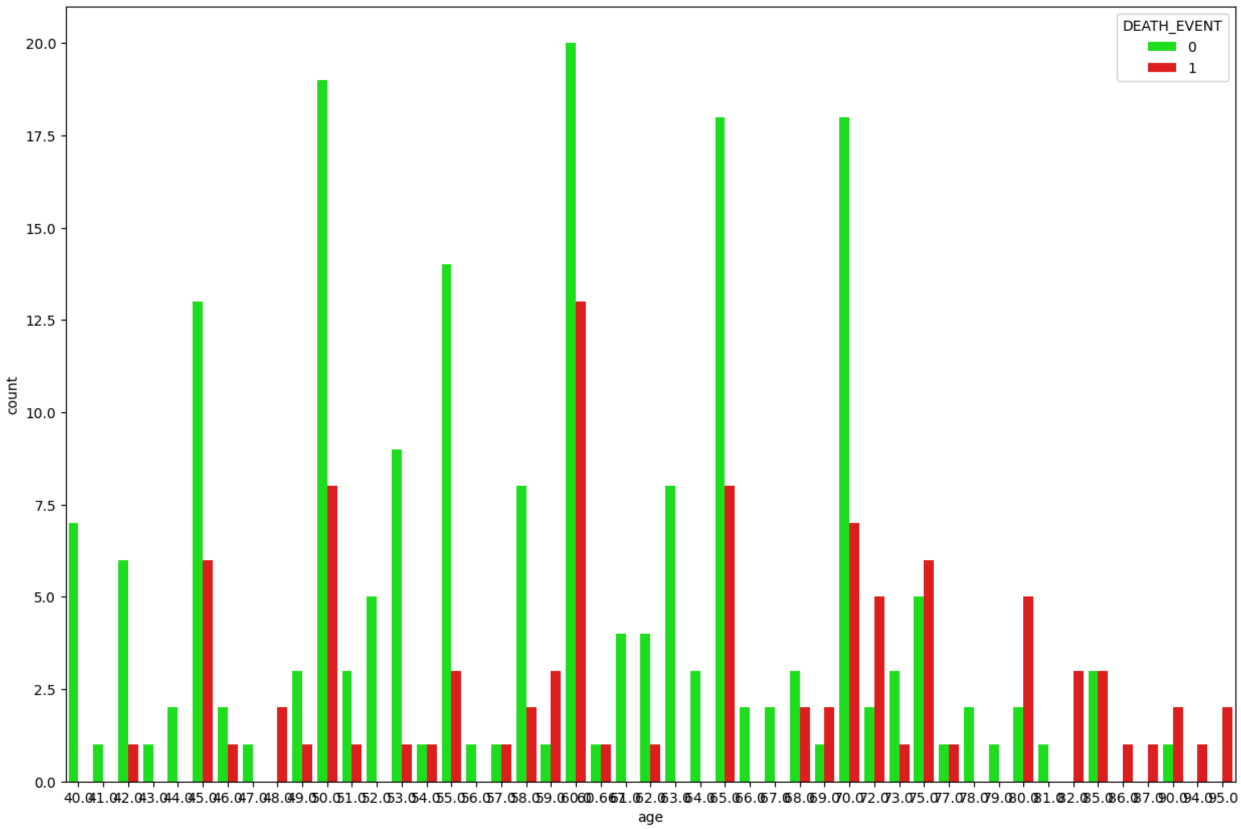
In [ ]:

```
# Assuming 'Sex' is the column containing 'M'
# Replace 'M' with 1 and 'F' with 0
data_df['sex'] = data_df['sex'].map({'M': 1, 'F': 0})

# Alternatively, if you want to exclude the column from correlation calculatio
numeric_df = data_df.select_dtypes(include=np.number)  # Select only numeric c
plt.subplots(figsize=(20, 20))
sns.heatmap(numeric_df.corr(), annot=True, cmap='viridis')
plt.show()
```



In [ ]:

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(15, 10))
cols = ["#00FF00", "#FF0000"]
days_of_the_week = sns.countplot(x=data_df['age'], data=data_df, hue='DEATH_EV
plt.show()
```

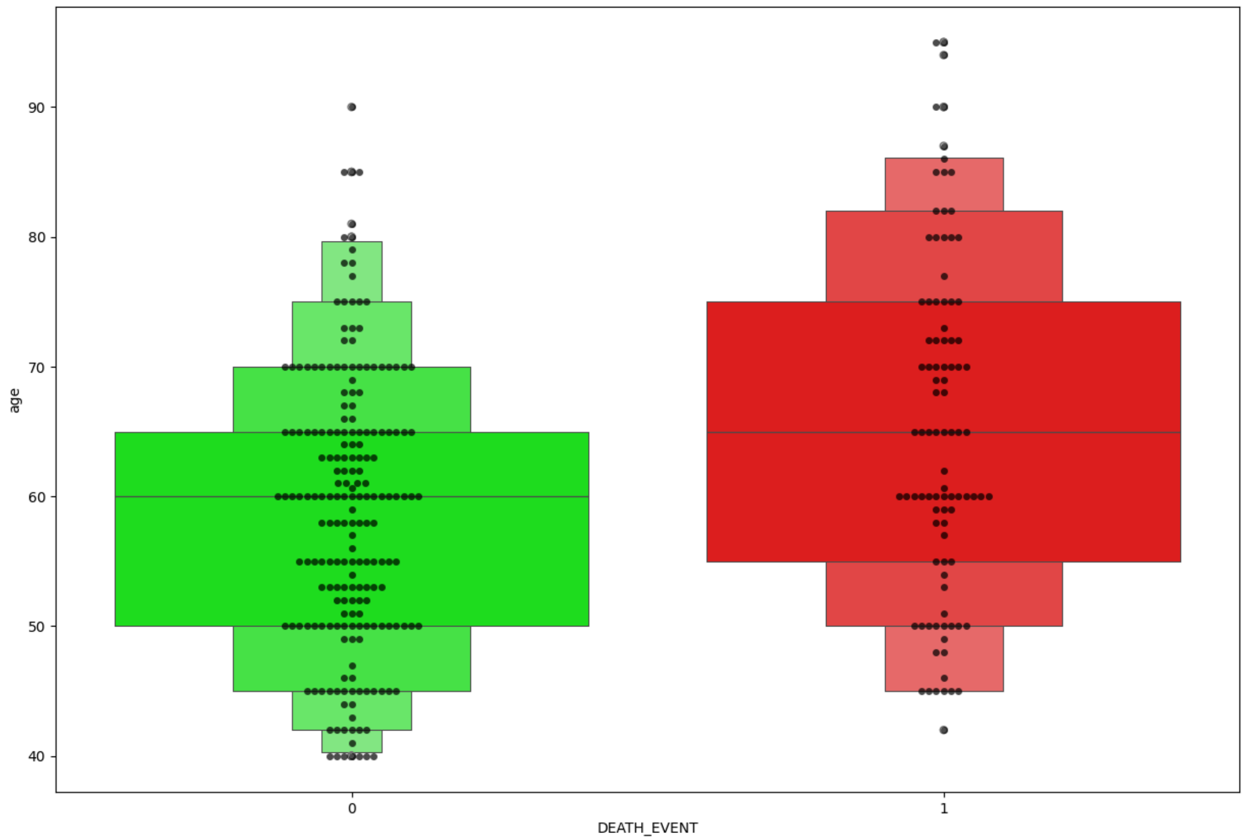


In [ ]:
```
print(data_df.columns)
```

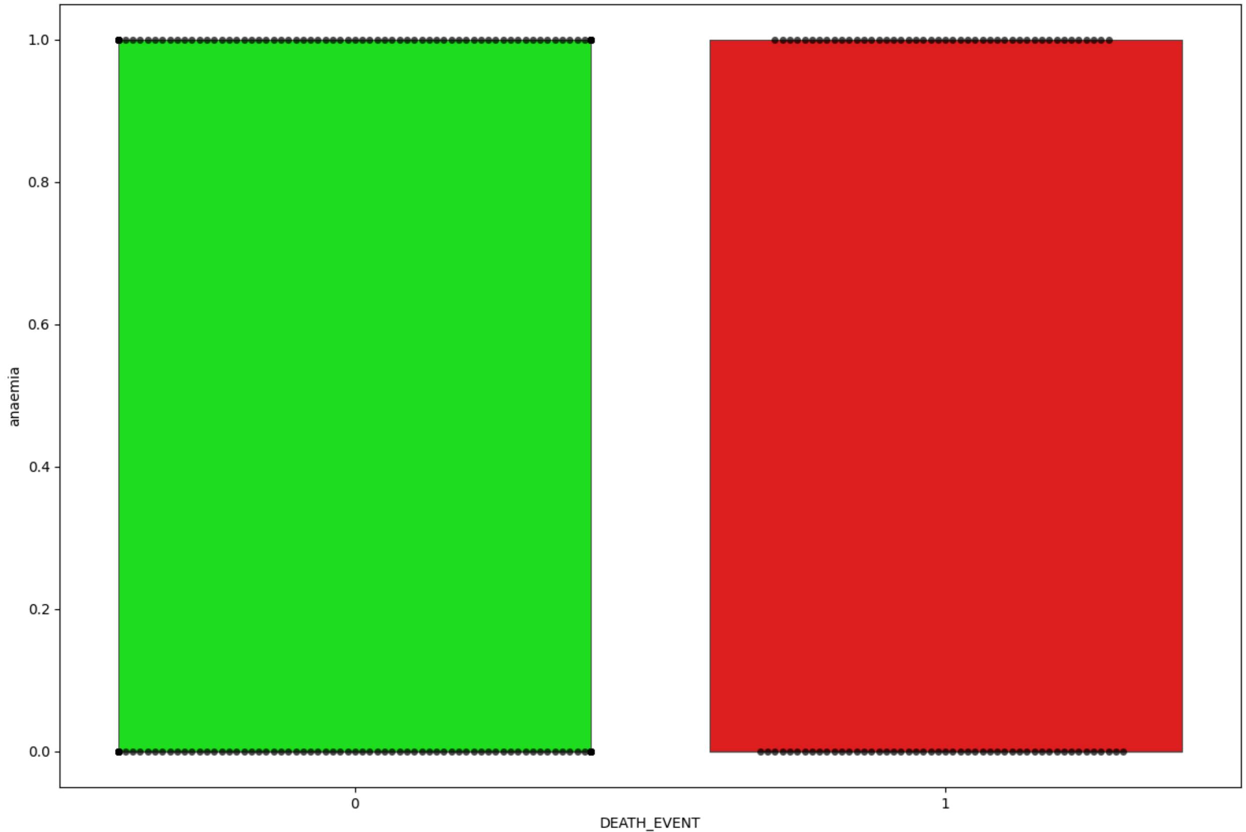
```
Index(['age', 'anaemia', 'creatinine_phosphokinase', 'diabetes',
       'ejection_fraction', 'high_blood_pressure', 'platelets',
       'serum_creatinine', 'serum_sodium', 'sex', 'smoking', 'time',
       'DEATH_EVENT'],
      dtype='object')
```

In [ ]:
```
feature = ['age', 'anaemia', 'creatinine_phosphokinase', 'diabetes',
       'ejection_fraction', 'high_blood_pressure', 'platelets',
       'serum_creatinine', 'serum_sodium', 'sex', 'smoking', 'time',
       'DEATH_EVENT']
for i in feature:
  plt.figure(figsize=(15,10))
  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()
```

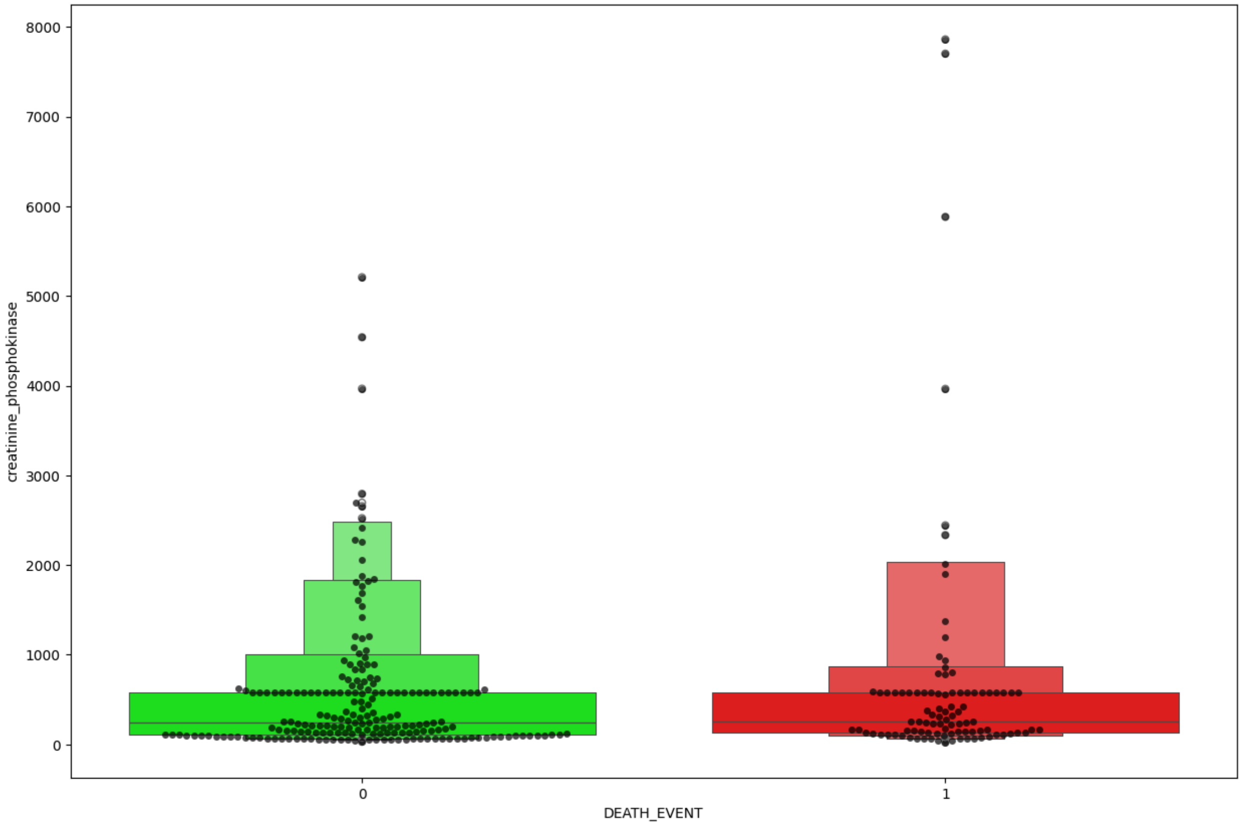
```
<ipython-input-277-8244dd70523e>:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same e
ffect.

  sns.boxenplot(x=data_df["DEATH_EVENT"],y=data_df[i],palette=cols)
```



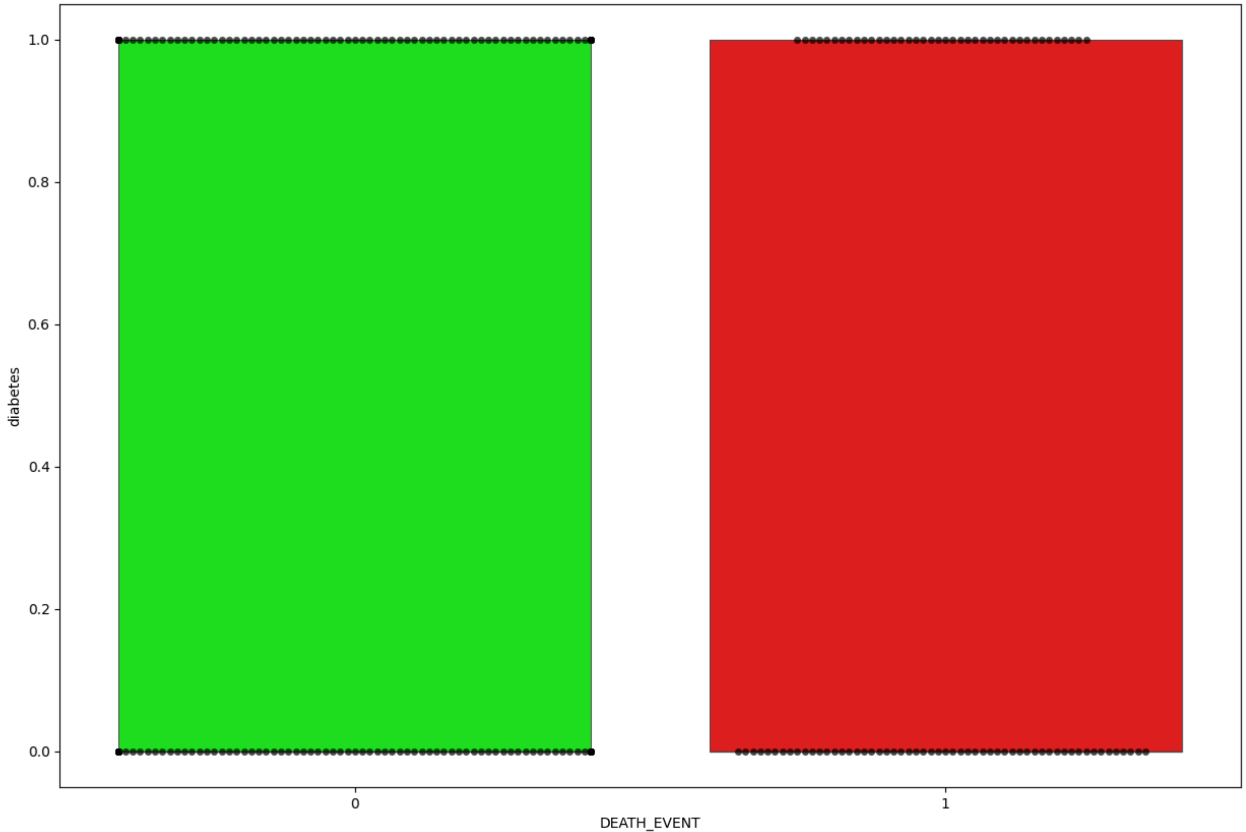

```
<ipython-input-277-8244dd70523e>:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same e
ffect.

  sns.boxenplot(x=data_df["DEATH_EVENT"],y=data_df[i],palette=cols)
/usr/local/lib/python3.11/dist-packages/seaborn/categorical.py:3399: UserWarnin
g: 37.9% of the points cannot be placed; you may want to decrease the size of t
he markers or use stripplot.
  warnings.warn(msg, UserWarning)
```



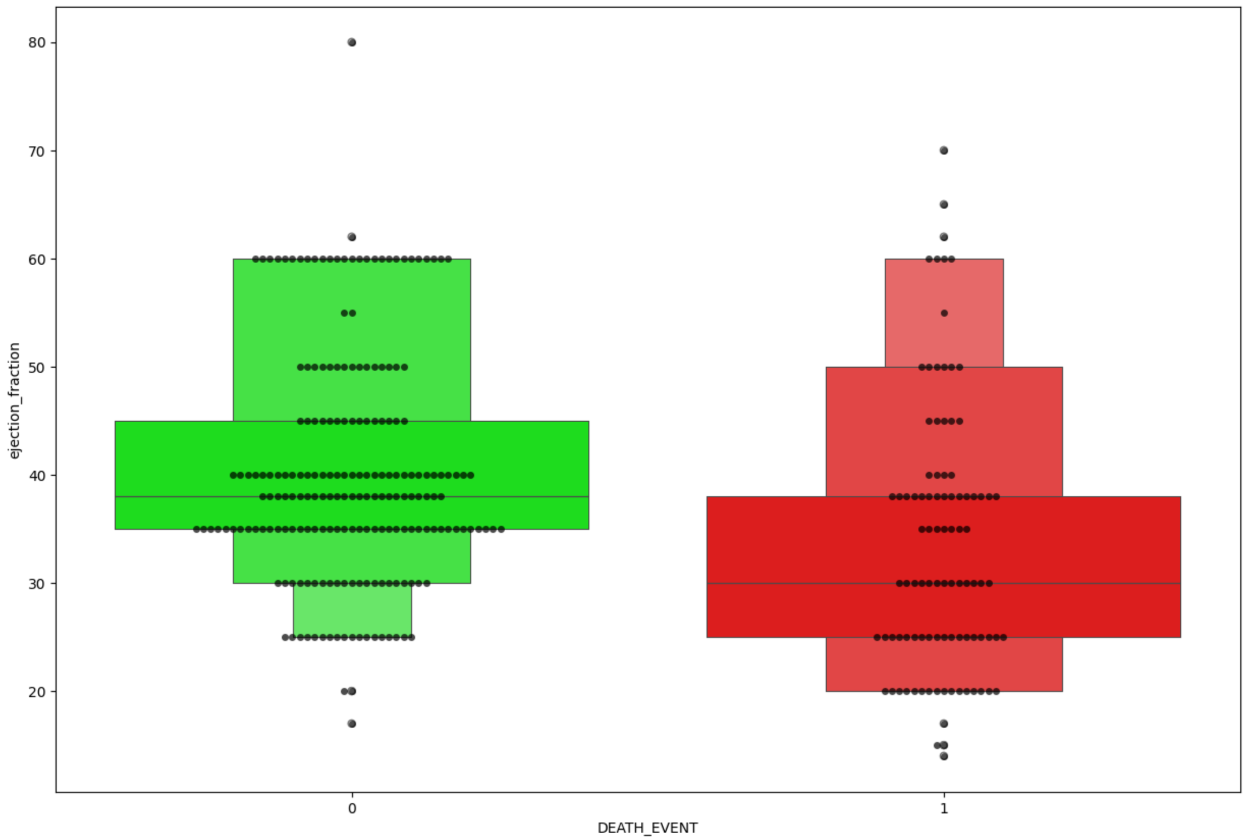

```
<ipython-input-277-8244dd70523e>:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same e
ffect.

  sns.boxenplot(x=data_df["DEATH_EVENT"],y=data_df[i],palette=cols)
```



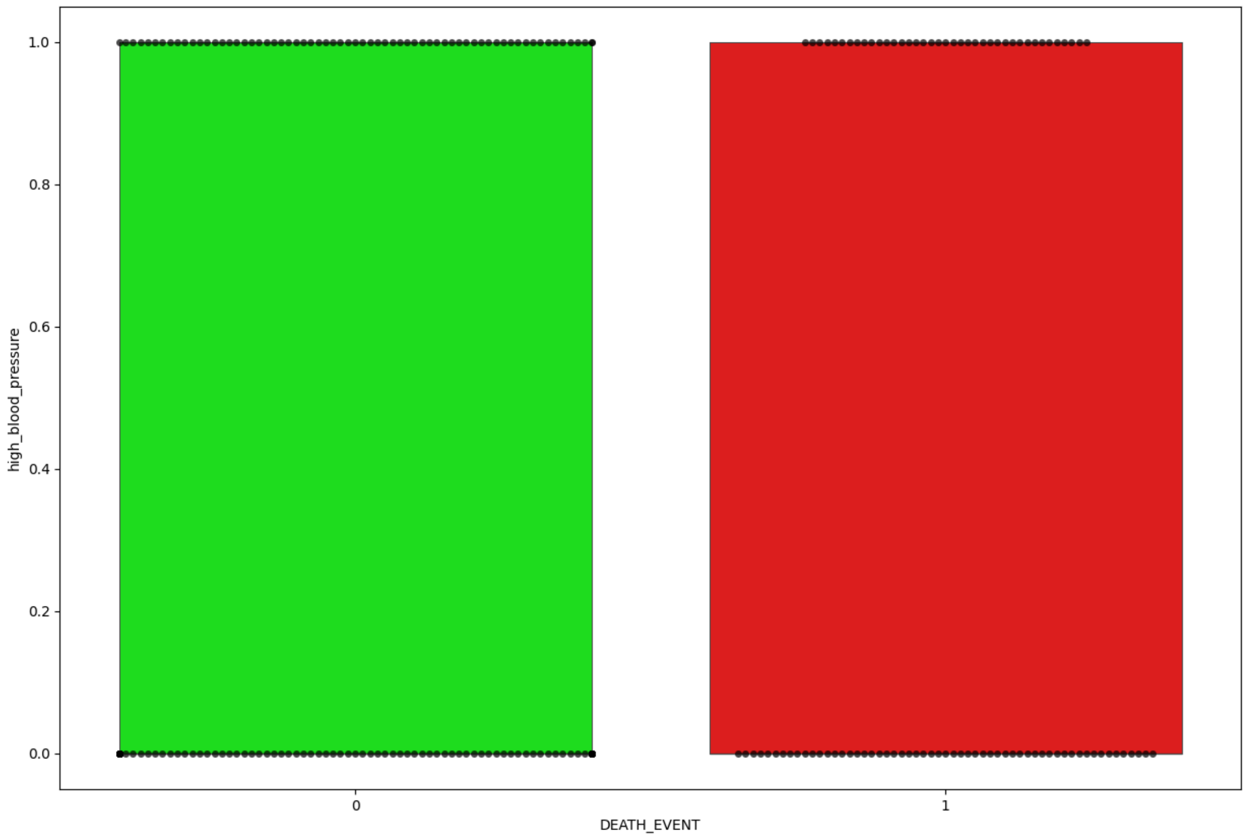

```
<ipython-input-277-8244dd70523e>:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same e
ffect.

  sns.boxenplot(x=data_df["DEATH_EVENT"],y=data_df[i],palette=cols)
/usr/local/lib/python3.11/dist-packages/seaborn/categorical.py:3399: UserWarnin
g: 37.9% of the points cannot be placed; you may want to decrease the size of t
he markers or use stripplot.
  warnings.warn(msg, UserWarning)
```



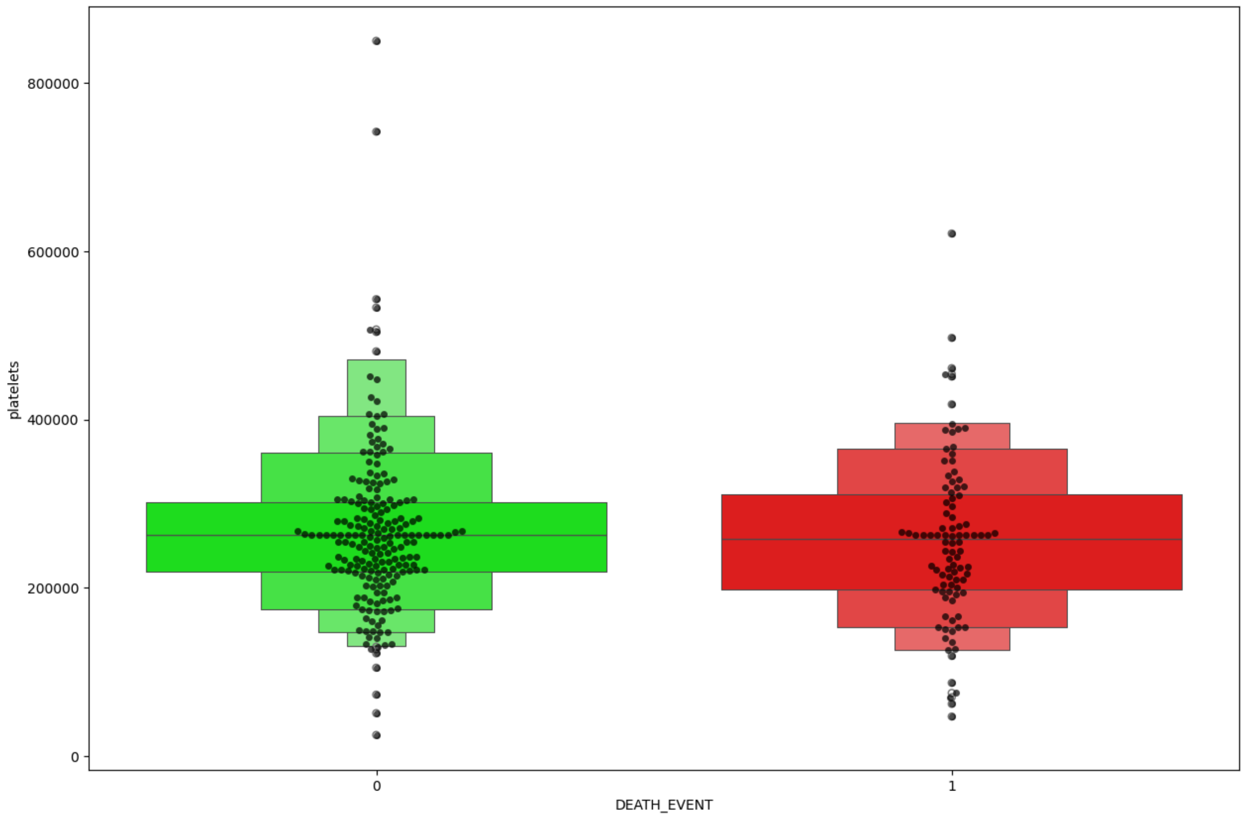

```
<ipython-input-277-8244dd70523e>:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same e
ffect.

  sns.boxenplot(x=data_df["DEATH_EVENT"],y=data_df[i],palette=cols)
```



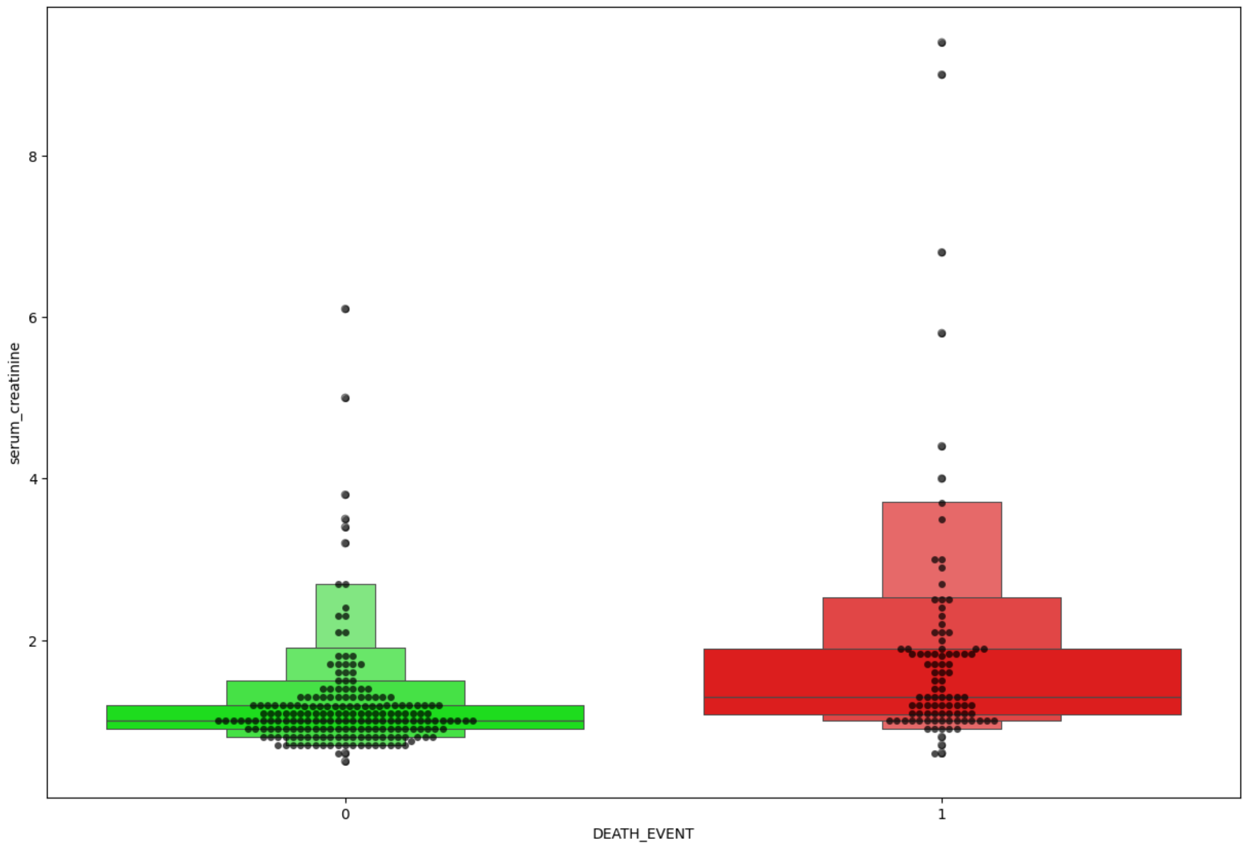

```
/usr/local/lib/python3.11/dist-packages/seaborn/categorical.py:3399: UserWarnin
g: 10.8% of the points cannot be placed; you may want to decrease the size of t
he markers or use stripplot.
  warnings.warn(msg, UserWarning)
<ipython-input-277-8244dd70523e>:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same e
ffect.

  sns.boxenplot(x=data_df["DEATH_EVENT"],y=data_df[i],palette=cols)
/usr/local/lib/python3.11/dist-packages/seaborn/categorical.py:3399: UserWarnin
g: 37.9% of the points cannot be placed; you may want to decrease the size of t
he markers or use stripplot.
  warnings.warn(msg, UserWarning)
```



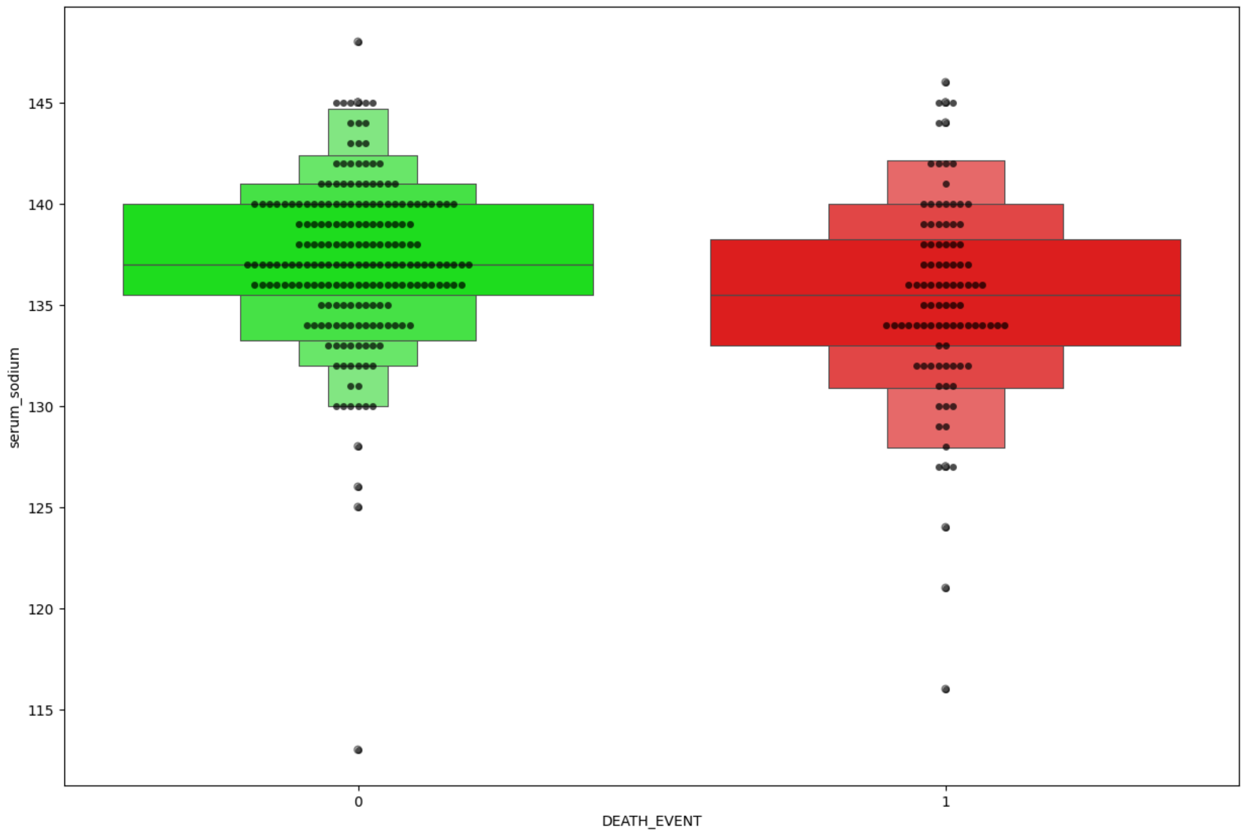

```
<ipython-input-277-8244dd70523e>:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same e
ffect.

  sns.boxenplot(x=data_df["DEATH_EVENT"],y=data_df[i],palette=cols)
```



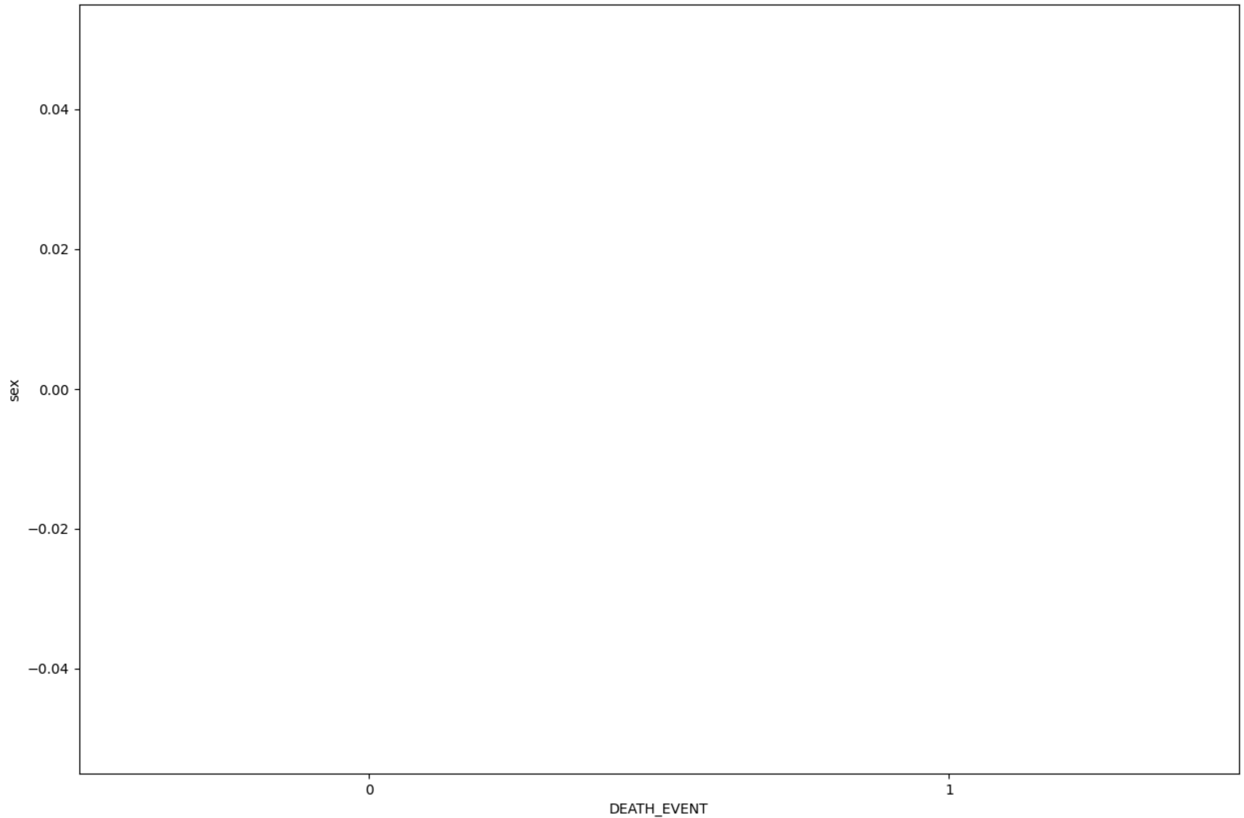

```
<ipython-input-277-8244dd70523e>:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxenplot(x=data_df["DEATH_EVENT"],y=data_df[i],palette=cols)
```



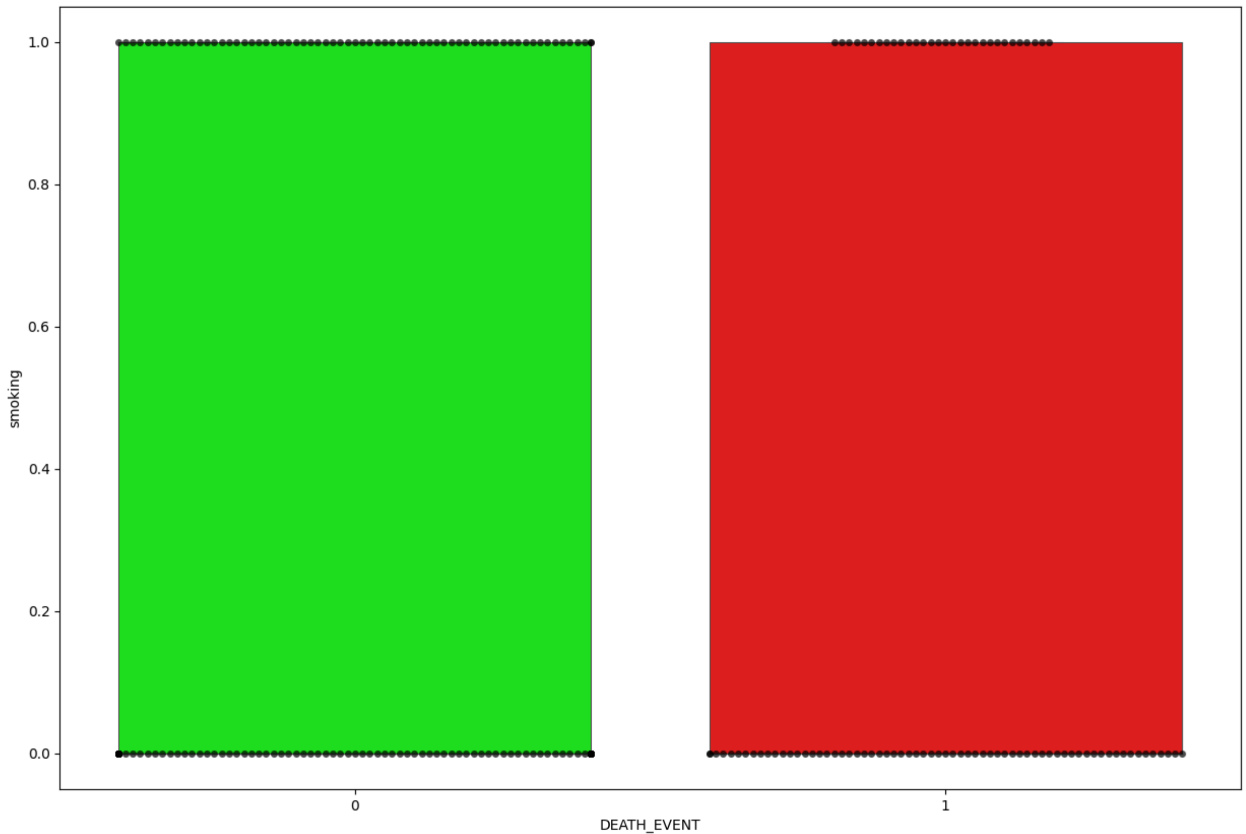

```
<ipython-input-277-8244dd70523e>:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same e
ffect.

  sns.boxenplot(x=data_df["DEATH_EVENT"],y=data_df[i],palette=cols)
```



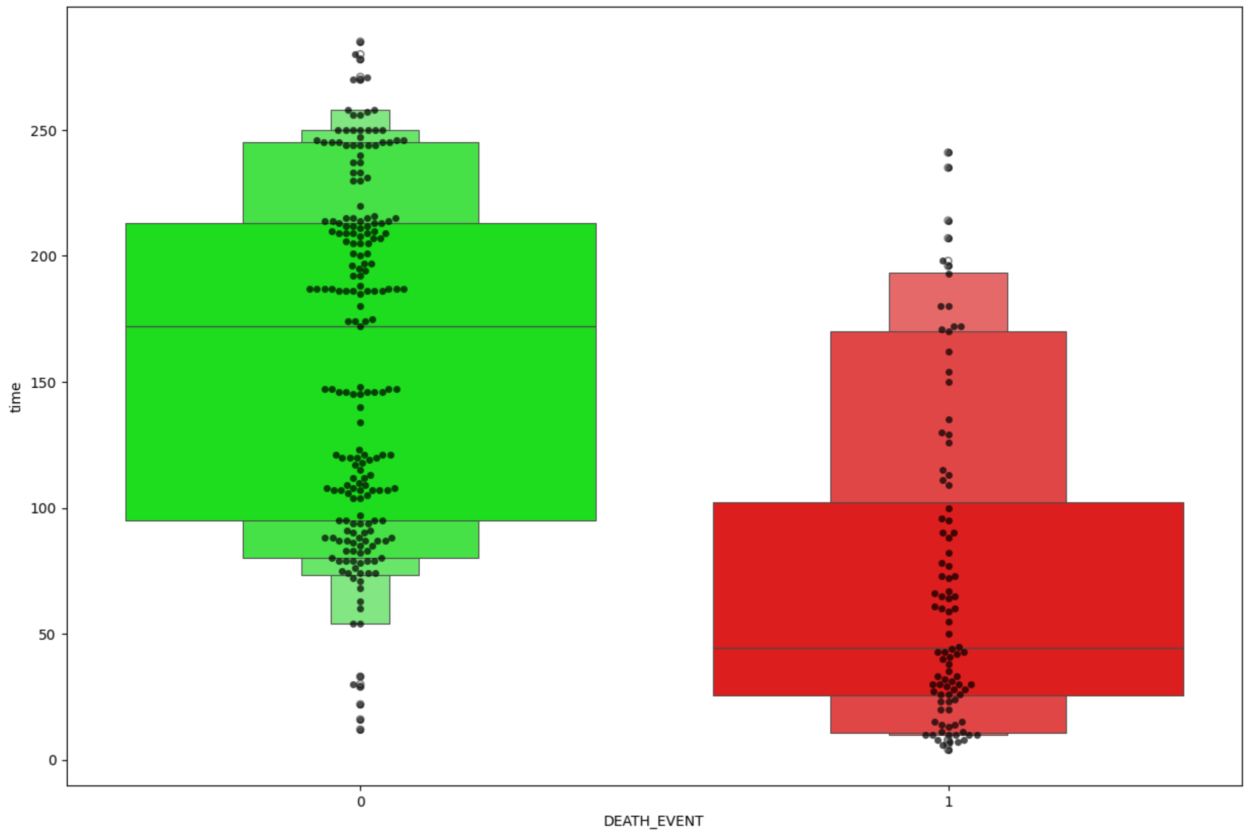

```
<ipython-input-277-8244dd70523e>:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same e
ffect.

  sns.boxenplot(x=data_df["DEATH_EVENT"],y=data_df[i],palette=cols)
```



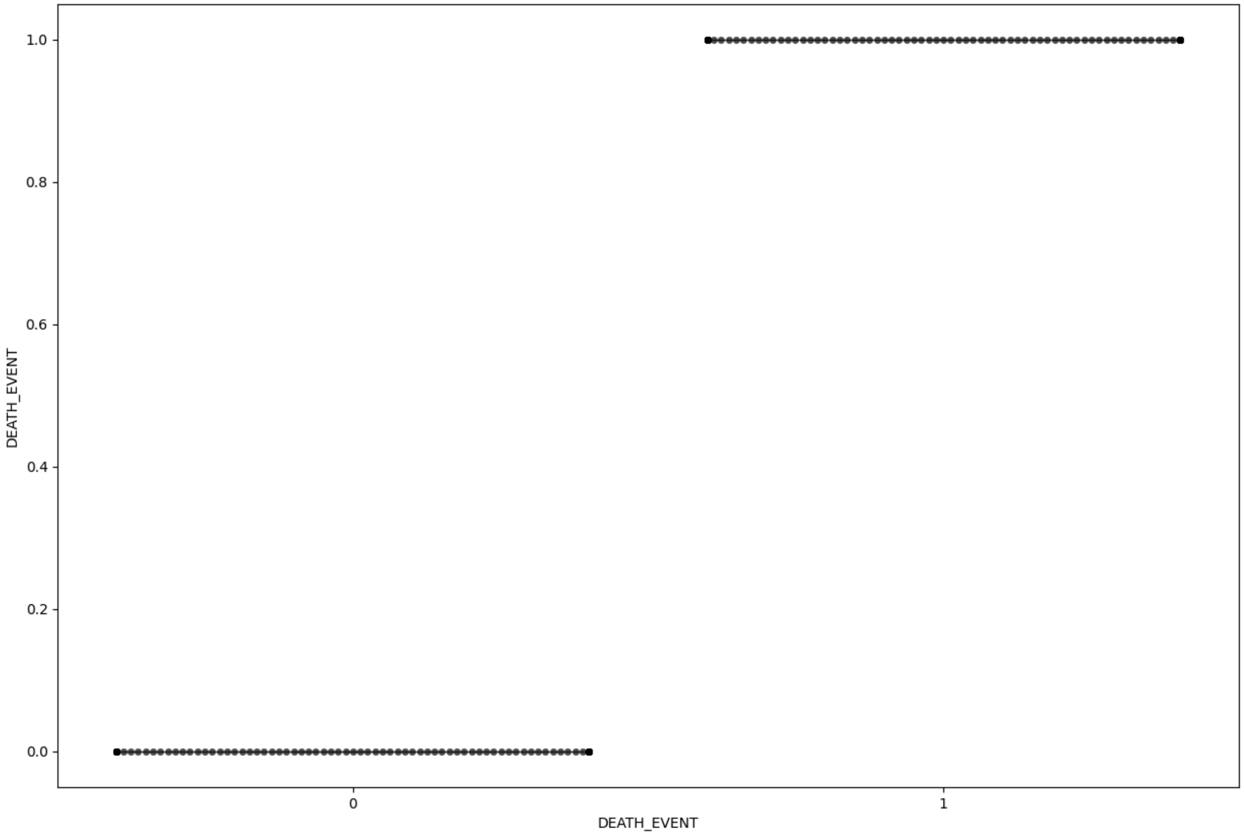

```
/usr/local/lib/python3.11/dist-packages/seaborn/categorical.py:3399: UserWarnin
g: 10.8% of the points cannot be placed; you may want to decrease the size of t
he markers or use stripplot.
  warnings.warn(msg, UserWarning)
<ipython-input-277-8244dd70523e>:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same e
ffect.

  sns.boxenplot(x=data_df["DEATH_EVENT"],y=data_df[i],palette=cols)
/usr/local/lib/python3.11/dist-packages/seaborn/categorical.py:3399: UserWarnin
g: 37.9% of the points cannot be placed; you may want to decrease the size of t
he markers or use stripplot.
  warnings.warn(msg, UserWarning)
```




```
<ipython-input-277-8244dd70523e>:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same e
ffect.

  sns.boxenplot(x=data_df["DEATH_EVENT"],y=data_df[i],palette=cols)
```




```
/usr/local/lib/python3.11/dist-packages/seaborn/categorical.py:3399: UserWarnin
g: 43.3% of the points cannot be placed; you may want to decrease the size of t
he markers or use stripplot.
  warnings.warn(msg, UserWarning)
<ipython-input-277-8244dd70523e>:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same e
ffect.

  sns.boxenplot(x=data_df["DEATH_EVENT"],y=data_df[i],palette=cols)
/usr/local/lib/python3.11/dist-packages/seaborn/categorical.py:3399: UserWarnin
g: 69.0% of the points cannot be placed; you may want to decrease the size of t
he markers or use stripplot.
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.11/dist-packages/seaborn/categorical.py:3399: UserWarnin
g: 34.4% of the points cannot be placed; you may want to decrease the size of t
he markers or use stripplot.
  warnings.warn(msg, UserWarning)
```



In [ ]:
```
# step 3 : Data processing model
```

In [ ]:
```
x=data_df.drop(columns='DEATH_EVENT',axis=1)
y=data_df['DEATH_EVENT']
```

In [ ]:
```
ss = StandardScaler()
x_scaled = ss.fit_transform(x)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/extmath.py:1101: RuntimeW
arning: invalid value encountered in divide
  updated_mean = (last_sum + new_sum) / updated_sample_count
/usr/local/lib/python3.11/dist-packages/sklearn/utils/extmath.py:1106: RuntimeW
arning: invalid value encountered in divide
  T = new_sum / new_sample_count
/usr/local/lib/python3.11/dist-packages/sklearn/utils/extmath.py:1126: RuntimeW
arning: invalid value encountered in divide
  new_unnormalized_variance -= correction**2 / new_sample_count
```

In [ ]:
```
x_train,x_test,y_train,y_test=train_test_split(x_scaled,y,test_size=0.3)
xs_train,xs_test,y_train,y_test = train_test_split(x_scaled,y,test_size = 0.3)
```

In [ ]:
```
len(xs_train)
```

Out[ ]: 209

In [ ]:
```
col_name = list(x.columns)
s_scaler = preprocessing.StandardScaler()
```

```
x_scaled = s_scaler.fit_transform(x)
x_scaled = pd.DataFrame(x_scaled, columns=col_name)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/extmath.py:1101: RuntimeW
arning: invalid value encountered in divide
  updated_mean = (last_sum + new_sum) / updated_sample_count
/usr/local/lib/python3.11/dist-packages/sklearn/utils/extmath.py:1106: RuntimeW
arning: invalid value encountered in divide
  T = new_sum / new_sample_count
/usr/local/lib/python3.11/dist-packages/sklearn/utils/extmath.py:1126: RuntimeW
arning: invalid value encountered in divide
  new_unnormalized_variance -= correction**2 / new_sample_count
```

In [ ]:

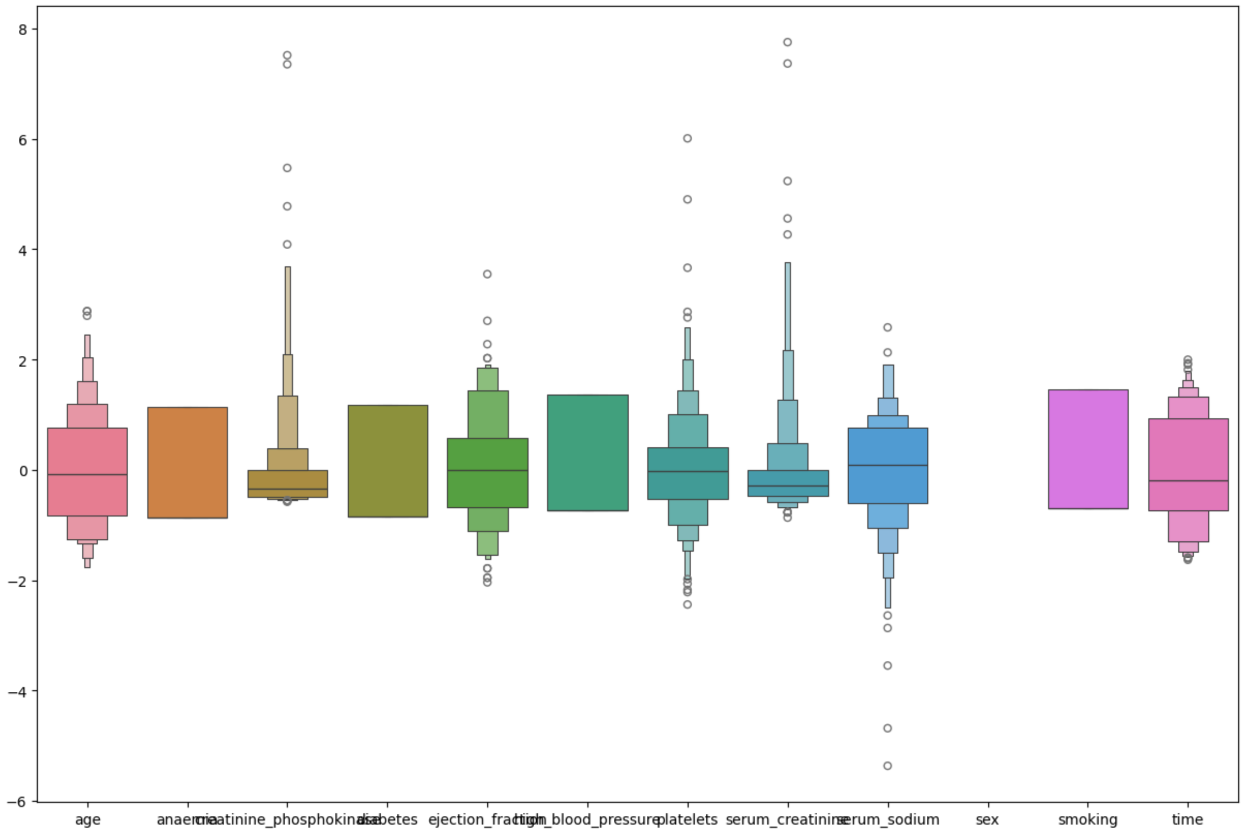
```
x_scaled.describe().T
```

Out[ ]:

	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	0.0	NaN	NaN	NaN	NaN	
smoking	299.0	-1.188199e-17	1.001676	-0.687682	-0.687682	-
time	299.0	-1.901118e-16	1.001676	-1.629502	-0.739000	-

In [ ]:

```
plt.figure(figsize=(15,10))
sns.boxenplot(data=x_scaled)
plt.show()
```



STEP 4 : ML/DL MODEL TESTING

In [ ]:
```
### This cell got error please don't run this cell

### model1=svm.SVC()
### model1.fit(x_train,y_train)

### y_pred=model1.predict(x_test)
### print(classification_report(y_test,y_pred))

### below is the correct cell and output
```

MODEL 1 AND MODEL 2

In [ ]:
```
import pandas as pd
import numpy as np
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.impute import SimpleImputer

# Assuming x_train, x_test, y_train, y_test are already loaded

# Ensure x_train and x_test are pandas DataFrames
x_train_df = pd.DataFrame(x_train)
x_test_df = pd.DataFrame(x_test)
```

```python
# Replace all occurrences of 9 with np.nan
x_train_df.replace(9, np.nan, inplace=True)
x_test_df.replace(9, np.nan, inplace=True)

# Initialize the SimpleImputer with the 'mean' strategy
imputer = SimpleImputer(strategy='mean')

# Fit the imputer on the training data and then transform both
x_train_imputed = imputer.fit_transform(x_train_df)
x_test_imputed = imputer.transform(x_test_df)

# ------------------- Model 1 (SVM) -------------------
model1 = svm.SVC()
model1.fit(x_train_imputed, y_train)
y_pred_model1 = model1.predict(x_test_imputed)
print("Classification Report for Model 1 (SVM):")
print(classification_report(y_test, y_pred_model1))
print("-" * 30)

# ------------------- Model 2 (RandomForestClassifier) -------------------
model2 = RandomForestClassifier(random_state=42) # Added random_state for repr
model2.fit(x_train_imputed, y_train)
y_pred_model2 = model2.predict(x_test_imputed)
print("Classification Report for Model 2 (RandomForestClassifier):")
print(classification_report(y_test, y_pred_model2))
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/impute/_base.py:635: UserWarnin
g: Skipping features without any observed values: [9]. At least one non-missing
value is needed for imputation with strategy='mean'.
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/impute/_base.py:635: UserWarnin
g: Skipping features without any observed values: [9]. At least one non-missing
value is needed for imputation with strategy='mean'.
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:156
5: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:156
5: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:156
5: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
Classification Report for Model 1 (SVM):
              precision    recall  f1-score   support

           0       0.67      1.00      0.80        60
           1       0.00      0.00      0.00        30

    accuracy                           0.67        90
   macro avg       0.33      0.50      0.40        90
weighted avg       0.44      0.67      0.53        90

------------------------------
Classification Report for Model 2 (RandomForestClassifier):
              precision    recall  f1-score   support

           0       0.67      0.97      0.79        60
           1       0.50      0.07      0.12        30

    accuracy                           0.67        90
   macro avg       0.59      0.52      0.46        90
weighted avg       0.62      0.67      0.57        90
```

FOR MODEL 1 AND MODEL 2 (Kneighbors Classifier)

In [ ]:

```
import pandas as pd
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.impute import SimpleImputer

# Assuming x_train, x_test, y_train, y_test are already loaded

# Ensure x_train and x_test are pandas DataFrames
x_train_df = pd.DataFrame(x_train)
x_test_df = pd.DataFrame(x_test)

# Replace all occurrences of 9 with np.nan
x_train_df.replace(9, np.nan, inplace=True)
x_test_df.replace(9, np.nan, inplace=True)

# Initialize the SimpleImputer with the 'mean' strategy
imputer = SimpleImputer(strategy='mean')

# Fit the imputer on the training data and then transform both
x_train_imputed = imputer.fit_transform(x_train_df)
x_test_imputed = imputer.transform(x_test_df)

# ------------------- Model 1 (KNeighborsClassifier) -------------------
model1 = KNeighborsClassifier(n_neighbors=5) # You can adjust the number of ne
model1.fit(x_train_imputed, y_train)
y_pred_model1 = model1.predict(x_test_imputed)
print("Classification Report for Model 1 (KNeighborsClassifier):")
```

```
print(classification_report(y_test, y_pred_model1))
print("-" * 30)

# ------------------- Model 2 (KNeighborsClassifier) -------------------
model2 = KNeighborsClassifier(n_neighbors=10) # You can use a different number
model2.fit(x_train_imputed, y_train)
y_pred_model2 = model2.predict(x_test_imputed)
print("Classification Report for Model 2 (KNeighborsClassifier):")
print(classification_report(y_test, y_pred_model2))
```

```
Classification Report for Model 1 (KNeighborsClassifier):
              precision    recall  f1-score   support

           0       0.63      0.77      0.69        60
           1       0.18      0.10      0.13        30

    accuracy                           0.54        90
   macro avg       0.40      0.43      0.41        90
weighted avg       0.48      0.54      0.50        90

------------------------------
Classification Report for Model 2 (KNeighborsClassifier):
              precision    recall  f1-score   support

           0       0.67      0.93      0.78        60
           1       0.33      0.07      0.11        30

    accuracy                           0.64        90
   macro avg       0.50      0.50      0.44        90
weighted avg       0.56      0.64      0.56        90
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/impute/_base.py:635: UserWarnin
g: Skipping features without any observed values: [9]. At least one non-missing
value is needed for imputation with strategy='mean'.
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/impute/_base.py:635: UserWarnin
g: Skipping features without any observed values: [9]. At least one non-missing
value is needed for imputation with strategy='mean'.
  warnings.warn(
```

In [ ]:

```
model = Sequential()

model.add(Dense(units = 64, activation = 'relu',input_dim = 12))
model.add(Dense(units = 16, activation = 'relu'))
model.add(Dense(units = 8,activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(units = 1, activation = 'relu'))

model.compile(optimizer='adam', loss = 'binary_crossentropy',metrics = ['accur

history = model.fit(x_train,y_train,batch_size=25 , epochs = 25, validation_sp
```

```
Epoch 1/25
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: User
Warning: Do not pass an `input_shape`/`input_dim` argument to a layer. When usi
ng Sequential models, prefer using an `Input(shape)` object as the first layer
in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 44ms/step - accuracy: 0.6980 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 2/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.7016 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 3/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6894 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 4/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6504 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 5/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.6629 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 6/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.7240 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 7/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.7031 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 8/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6576 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 9/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6580 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 10/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.7065 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 11/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6565 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 12/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6932 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 13/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6793 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 14/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6478 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 15/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.7006 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 16/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.6828 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 17/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.7176 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 18/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.6617 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 19/25
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.6940 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 20/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.7015 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 21/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.7034 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 22/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.6779 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 23/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.6848 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 24/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.6900 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 25/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.6715 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
```

In [ ]:

```
model = Sequential()

model.add(Dense(units = 64, activation = 'relu',input_dim = 12))
model.add(Dense(units = 16, activation = 'relu'))
model.add(Dense(units = 8,activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(units = 1, activation = 'relu'))

model.compile(optimizer='adam', loss = 'binary_crossentropy',metrics = ['accur

history = model.fit(xs_train,y_train,batch_size=25 , epochs = 25, validation_s
```

```
Epoch 1/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 3s 46ms/step - accuracy: 0.6476 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 2/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6497 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 3/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6629 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 4/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 0.6921 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 5/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6920 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 6/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6833 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 7/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6765 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 8/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6410 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 9/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6919 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 10/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6635 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 11/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.6785 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 12/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.7165 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 13/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.7051 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 14/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.7032 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 15/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.6637 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 16/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.6576 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 17/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6709 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 18/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.6665 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
```

```
Epoch 19/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6903 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 20/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.6932 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 21/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 0.6612 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 22/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6890 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 23/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6835 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 24/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6955 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
Epoch 25/25
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6786 - loss: nan - val_accu
racy: 0.6981 - val_loss: nan
```

In [ ]:
```
y_pred = model.predict(xs_test)
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
```

In [ ]:
```
y_pred
```

```
Out[ ]: array([[nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
               [nan],
```

```
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan]], dtype=float32)
```

In [ ]:
```
x_train
```

Out[ ]:
```
array([[ 0.0981993 , -0.87110478, -0.56969133, ...,         nan,
        -0.68768191, -0.17114264],
       [-0.15443437, -0.87110478, -0.53252674, ...,         nan,
        -0.68768191,  0.06116245],
       [-0.65970173, -0.87110478, -0.38696543, ...,         nan,
        -0.68768191,  1.08072371],
       ...,
       [ 2.03505748, -0.87110478, -0.46748871, ...,         nan,
         1.4541607 , -0.51960029],
       [-0.74391295, -0.87110478, -0.46439166, ...,         nan,
         1.4541607 , -0.23567184],
       [ 0.35083298,  1.14796753, -0.46129461, ...,         nan,
        -0.68768191,  0.82260694]])
```

In [ ]:
```
x_test.shape
```

Out[ ]:
```
(90, 12)
```

In [ ]:
```
import numpy as np

print(np.isnan(x_test).any())  # Returns True if there is at least one NaN
print(np.sum(np.isnan(x_test))) # Returns the total number of NaNs
print(np.where(np.isnan(x_test))) # Returns the indices where NaNs are locatec
```

```
True
90
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
       51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
       68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
       85, 86, 87, 88, 89]), array([9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9,
9, 9, 9, 9, 9, 9, 9, 9,
       9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9,
       9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9,
       9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9,
       9, 9]))
```

In [ ]:
```
y_test
```

Out[ ]:

	DEATH_EVENT
248	0
298	0
252	0
65	1
290	0
...	...
83	0
34	1
41	1
116	0
9	1

90 rows × 1 columns

dtype: int64

In [ ]:
```
import pandas as pd
```

```
# Assuming x_test is a NumPy array, convert it to a DataFrame for easier inspe
x_test_df = pd.DataFrame(x_test)
print(x_test_df.isnull().sum())
```

```
0      0
1      0
2      0
3      0
4      0
5      0
6      0
7      0
8      0
9     90
10     0
11     0
dtype: int64
```

In [ ]:
```
# 2.ANN
```

In [ ]:
```
early_stopping = callbacks.EarlyStopping(
min_delta = 0.001,patience=20,restore_best_weights=True
)

model = Sequential()

model.add(Dense(units=16,kernel_initializer='uniform',activation='relu',input_
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'))
```

In [ ]:
```
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy']
```

In [ ]:
```
model.summary()
```

Model: "sequential_22"

Layer (type)	Output Shape	Param #
dense_78 (Dense)	(None, 16)	208
dense_79 (Dense)	(None, 8)	136
dropout_21 (Dropout)	(None, 8)	0
dense_80 (Dense)	(None, 8)	72
dropout_22 (Dropout)	(None, 8)	0
dense_81 (Dense)	(None, 1)	9

Total params: 425 (1.66 KB)

Trainable params: 425 (1.66 KB)

Non-trainable params: 0 (0.00 B)

In [ ]:
```
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
y_train shape: (209,)
y_test shape: (90,)
```

In [ ]:
```
import numpy as np

y_train = np.array(y_train).reshape(-1, 1)
y_test = np.array(y_test).reshape(-1, 1)

print("Reshaped y_train shape:", y_train.shape)
print("Reshaped y_test shape:", y_test.shape)
```

```
Reshaped y_train shape: (209, 1)
Reshaped y_test shape: (90, 1)
```

In [ ]:
```
# Assuming you have already loaded and preprocessed your x_train data
number_of_features = x_train.shape[1]

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
    Dense(128, activation='relu', input_shape=(number_of_features,)),
    Dense(64, activation='tanh'),
    Dense(1, activation='sigmoid') # Output layer for binary classification
])

# Now you can compile and fit your model
optimizer = 'adam'
loss = 'binary_crossentropy'
metrics = ['accuracy']
model.compile(optimizer=optimizer, loss=loss, metrics=metrics)

# ... your model.fit() code ...
```

In [ ]:
```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy
```

In [ ]:
```
from tensorflow.keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_w

history = model.fit(
    x_train,
    y_train,
    batch_size=25,  # Adjust as needed
    epochs=100,     # Adjust as needed
    validation_data=(x_test, y_test), # If you have test data for validation
```

```
    callbacks=[early_stopping],
    validation_split = 0.25
)
```

```
Epoch 1/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 2s 34ms/step - accuracy: 0.6723 - loss: nan - val_accu
racy: 0.6667 - val_loss: nan
Epoch 2/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.7116 - loss: nan - val_accu
racy: 0.6667 - val_loss: nan
Epoch 3/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.7057 - loss: nan - val_accu
racy: 0.6667 - val_loss: nan
Epoch 4/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6779 - loss: nan - val_accu
racy: 0.6667 - val_loss: nan
Epoch 5/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6913 - loss: nan - val_accu
racy: 0.6667 - val_loss: nan
Epoch 6/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6863 - loss: nan - val_accu
racy: 0.6667 - val_loss: nan
Epoch 7/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6747 - loss: nan - val_accu
racy: 0.6667 - val_loss: nan
Epoch 8/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6824 - loss: nan - val_accu
racy: 0.6667 - val_loss: nan
Epoch 9/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6661 - loss: nan - val_accu
racy: 0.6667 - val_loss: nan
Epoch 10/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.7366 - loss: nan - val_accu
racy: 0.6667 - val_loss: nan
Epoch 11/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.7163 - loss: nan - val_accu
racy: 0.6667 - val_loss: nan
```

In [ ]:

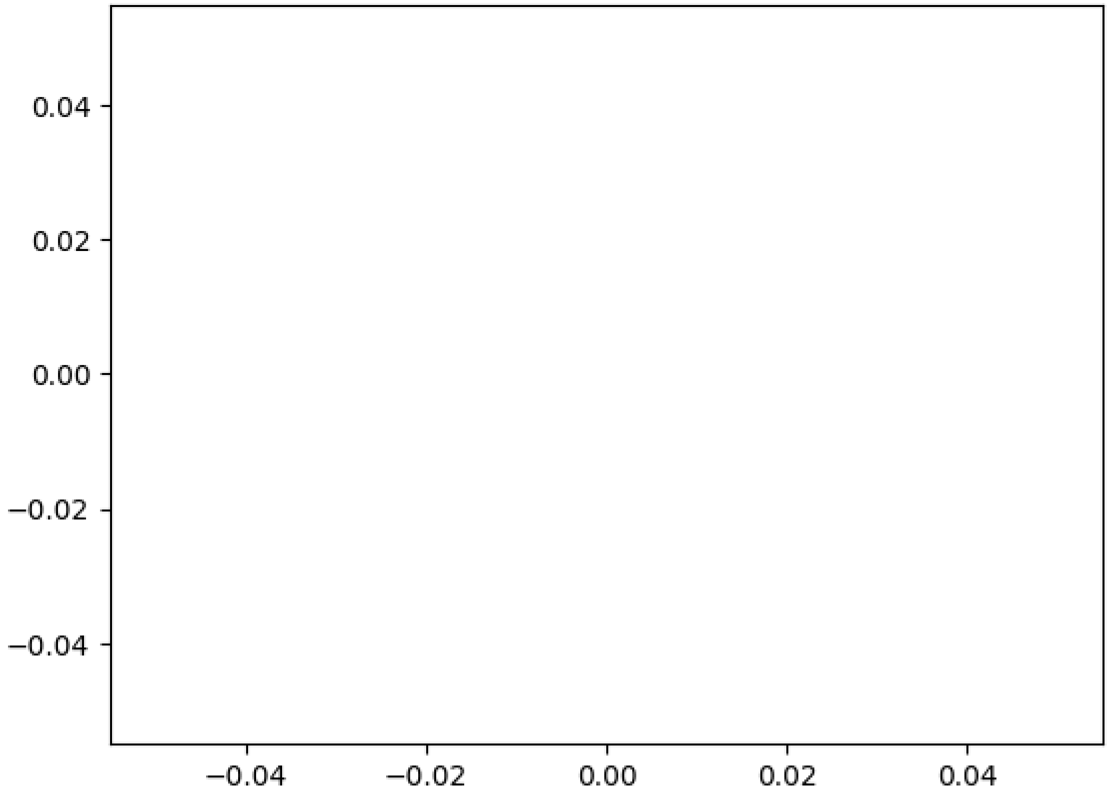
```
print(history.history.keys())
for key in history.history.keys():
    print(f"Shape of '{key}': {len(history.history[key])}, First few values: {
```

```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
Shape of 'accuracy': 11, First few values: [0.6842105388641357, 0.6842105388641
357, 0.6842105388641357, 0.6842105388641357, 0.6842105388641357]
Shape of 'loss': 11, First few values: [nan, nan, nan, nan, nan]
Shape of 'val_accuracy': 11, First few values: [0.6666666865348816, 0.666666686
5348816, 0.6666666865348816, 0.6666666865348816, 0.6666666865348816]
Shape of 'val_loss': 11, First few values: [nan, nan, nan, nan, nan]
```

In [ ]:

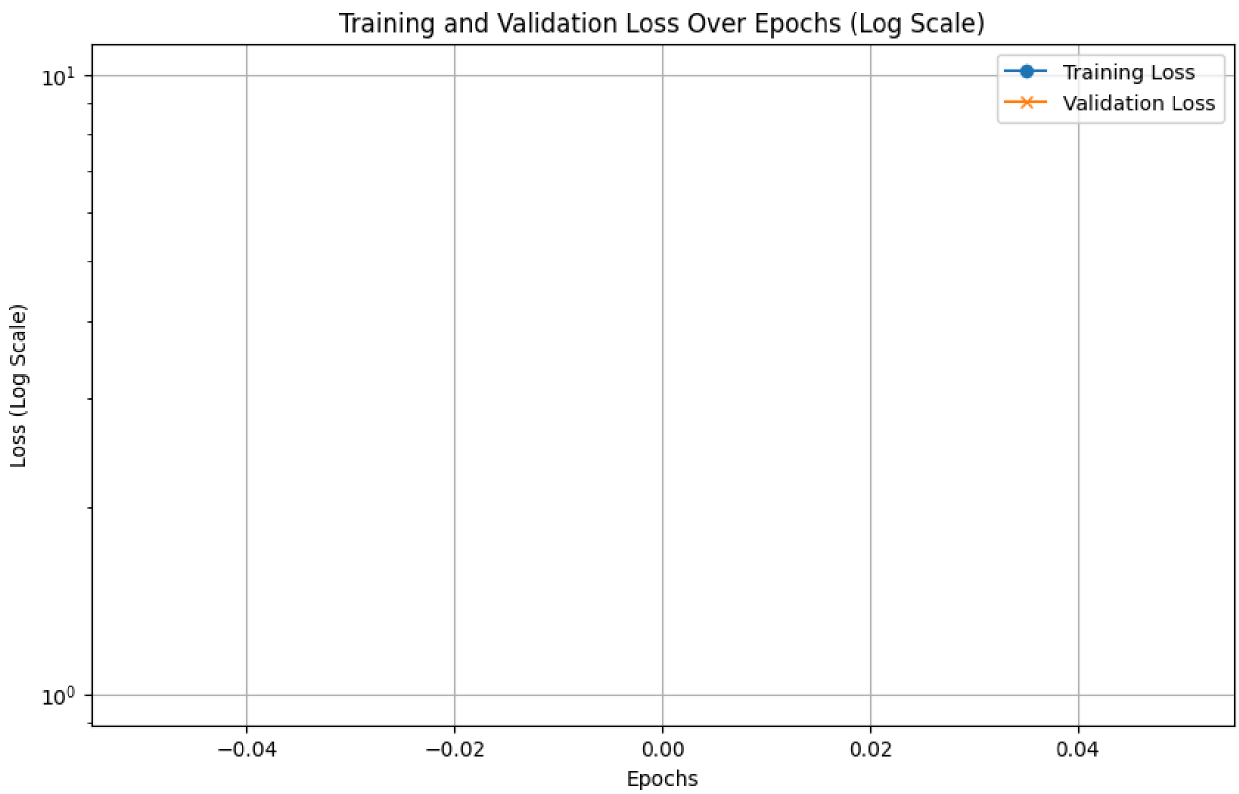
```
import pandas as pd
import matplotlib.pyplot as plt

# Assuming 'history' is the result of model.fit()
```

```
history_df = pd.DataFrame(history.history)

plt.plot(history_df.loc[:, ['loss']], label="Training Loss")
plt.plot(history_df.loc[:, ['val_loss']], label="Validation Loss")
plt.show()
```

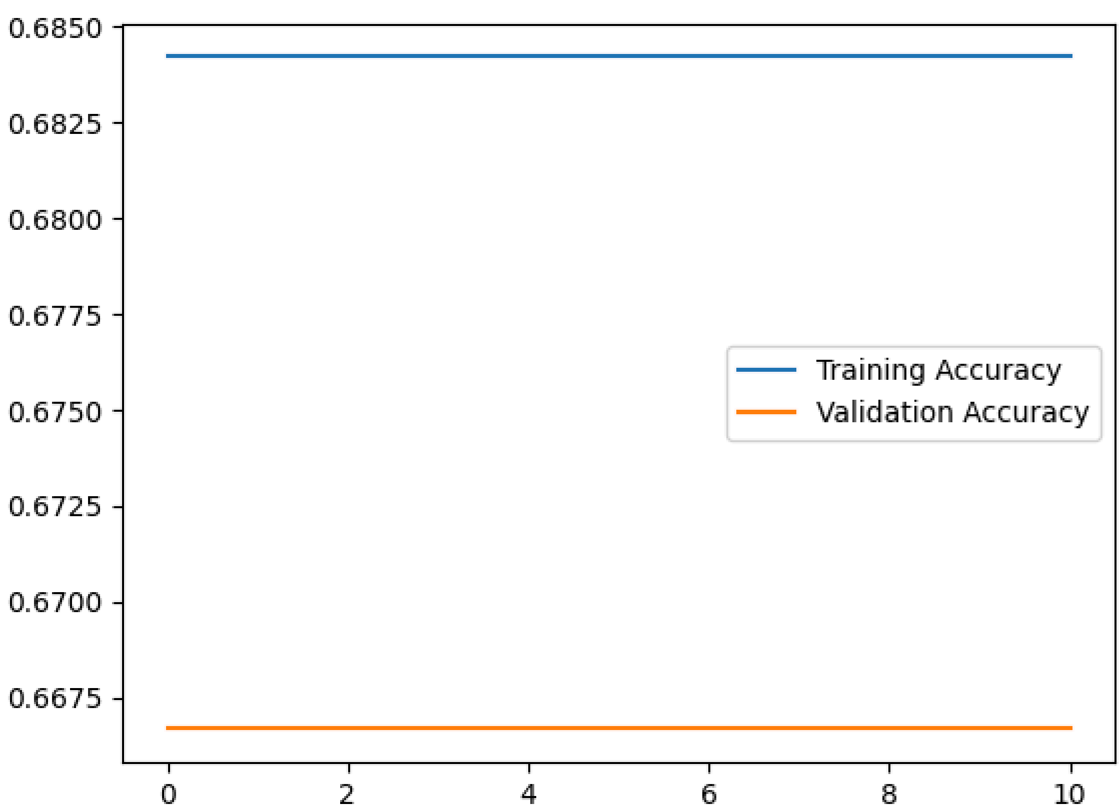


In [ ]:

```
import pandas as pd
import matplotlib.pyplot as plt

history_df = pd.DataFrame(history.history)
epochs = range(1, len(history_df) + 1)

plt.figure(figsize=(10, 6))
plt.plot(epochs, history_df['loss'], label="Training Loss", marker='o')
plt.plot(epochs, history_df['val_loss'], label="Validation Loss", marker='x')
plt.xlabel("Epochs")
plt.ylabel("Loss (Log Scale)")
plt.title("Training and Validation Loss Over Epochs (Log Scale)")
plt.yscale('log')  # Apply logarithmic scale to the y-axis
plt.legend()
plt.grid(True)
plt.show()
```

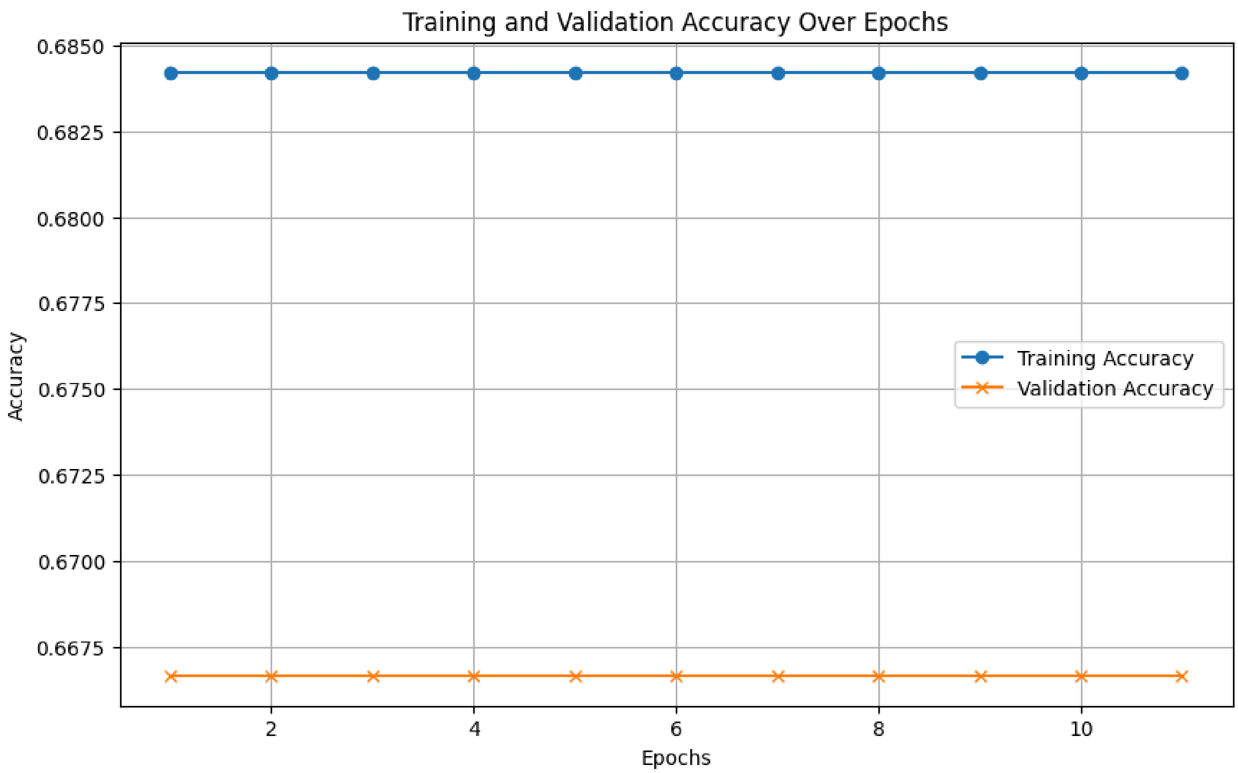


In [ ]:
```
plt.plot(history_df['accuracy'], label='Training Accuracy')
plt.plot(history_df['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.show()
```



In [ ]:

```
plt.figure(figsize=(10, 6))
plt.plot(epochs, history_df['accuracy'], label="Training Accuracy", marker='o'
plt.plot(epochs, history_df['val_accuracy'], label="Validation Accuracy", mark
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Training and Validation Accuracy Over Epochs")
plt.legend()
plt.grid(True)
plt.show()
```



In [ ]:

```
print("Training Loss:", history.history['loss'])
print("Validation Loss:", history.history['val_loss'])
```

```
Training Loss: [nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan]
Validation Loss: [nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan]
```

In [ ]:

```
print("Training Accuracy:", history.history['accuracy'])
print("Validation Accuracy:", history.history['val_accuracy'])
```

```
Training Accuracy: [0.6842105388641357, 0.6842105388641357, 0.6842105388641357,
0.6842105388641357, 0.6842105388641357, 0.6842105388641357, 0.6842105388641357,
0.6842105388641357, 0.6842105388641357, 0.6842105388641357, 0.6842105388641357]
Validation Accuracy: [0.6666666865348816, 0.6666666865348816, 0.666666686534881
6, 0.6666666865348816, 0.6666666865348816, 0.6666666865348816, 0.66666668653488
16, 0.6666666865348816, 0.6666666865348816, 0.6666666865348816, 0.6666666865348
816]
```

In [ ]:

```
y_pred = model.predict(x_test)
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
```

In [ ]:
```
y_pred = (y_pred > 0.5)
```

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

```
              precision    recall  f1-score   support

           0       0.67      1.00      0.80        60
           1       0.00      0.00      0.00        30

    accuracy                           0.67        90
   macro avg       0.33      0.50      0.40        90
weighted avg       0.44      0.67      0.53        90
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:156
5: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:156
5: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:156
5: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: