2| IMPORT NECESSARY LIBRARIES

In [1]:

```python
import pandas as pd  #for data manipulation operations
import numpy as np  #for numeric operations on data
import seaborn as sns  #for data visualization operations
import matplotlib.pyplot as plt  #for data visualization operations
from sklearn.preprocessing import LabelEncoder # for encoding
from sklearn.preprocessing import StandardScaler #for standardization

from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.metrics import roc_auc_score,roc_curve
from sklearn.metrics import plot_confusion_matrix
from sklearn import model_selection
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from termcolor import colored
#!pip install xgboost
from xgboost import XGBRegressor

#!pip install lightgbm
from lightgbm import LGBMRegressor

#ignore warnings
import warnings
warnings.filterwarnings("ignore")

from sklearn import set_config
set_config(print_changed_only = False)

sns.set_theme(style = "whitegrid")

print(colored("\n THE REQUIRED LIBRARIES WERE SUCCESFULLY IMPORTED...", color = "green", attrs = ["bold", "dark"]))
```

```
THE REQUIRED LIBRARIES WERE SUCCESFULLY IMPORTED...
```

3 LOAD DATASET

In [2]:

```
heart = pd.read_csv("../input/heart-failure-clinical-data/heart_failure_clinical_records_dataset.csv")
df = heart.copy()
df.head(n = 10).style.background_gradient(cmap = "Reds_r").set_properties(**{"font-family" : "Segoe UI"}).hide_index()
```

Out[2]:

age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure
75.000000	0	582	0	20	1
55.000000	0	7861	0	38	0
65.000000	0	146	0	20	0
50.000000	1	111	0	20	0
65.000000	1	160	1	20	0
90.000000	1	47	0	40	1
75.000000	1	246	0	15	0
60.000000	1	315	1	60	0
65.000000	0	157	0	65	0
80.000000	1	123	0	35	1

4 | INITIAL INFORMATION ABOUT DATASET

4.1 | Get Initial Information

In [3]:

```
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
```

Brief information

The dataset consists of **299 rows and 13 columns.** The data type of all variables are **numeric.**

4.2 | Change Column Values

In [4]:

```
df_eda = pd.DataFrame()

df_eda["age"] = df["age"]
df_eda["anaemia"] = np.where(df["anaemia"] < 1, "no", "yes")
df_eda["creatinine_phosphokinase"] = df["creatinine_phosphokinase"]
df_eda["diabetes"] = np.where(df["diabetes"] < 1, "no", "yes")
df_eda["ejection_fraction"] = df["ejection_fraction"]
df_eda["high_blood_pressure"] = np.where(df["high_blood_pressure"] < 1, "no", "yes")
df_eda["platelets"] = df["platelets"]
df_eda["serum_creatinine"] = df["serum_creatinine"]
df_eda["serum_sodium"] = df["serum_sodium"]
df_eda["sex"] = np.where(df["sex"] < 1, "female", "male")
df_eda["smoking"] = np.where(df["smoking"] < 1, "no", "yes")
df_eda["time"] = df["time"]
df_eda["death_event"] = np.where(df["DEATH_EVENT"] < 1, "no", "yes")

df_eda.head().style.background_gradient(cmap = "Reds").set_properties(**{"font-family" : "Segoe UI"}).hide_index()
```

Out[4]:

age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure
75.000000	no	582	no	20	yes
55.000000	no	7861	no	38	no
65.000000	no	146	no	20	no
50.000000	yes	111	no	20	no
65.000000	yes	160	yes	20	no

4.3 | Descriptive Statistics of Numeric Variables

In [5]:

```python
df_eda.describe().T.style.background_gradient(cmap = "Reds_r").set_properties(**{"font-family" : "Segoe UI"})
```

Out[5]:

	count	mean	std	min	25%
age	299.000000	60.833893	11.894809	40.000000	51.000000
creatinine_phosphokinase	299.000000	581.839465	970.287881	23.000000	116.500000
ejection_fraction	299.000000	38.083612	11.834841	14.000000	30.000000
platelets	299.000000	263358.029264	97804.236869	25100.000000	212500.000000
serum_creatinine	299.000000	1.393880	1.034510	0.500000	0.900000
serum_sodium	299.000000	136.625418	4.412477	113.000000	134.000000
time	299.000000	130.260870	77.614208	4.000000	73.000000

Basic descriptive statistics

- **The average value of age is 60.83 , the highest value is 95**
- **The average value of creatinine_phosphokinase is 581.83 , the highest value is 7861**
- **The average value of ejection_fraction is 30.08 , the highest value is 80**
- **The average value of platelets is 263358 , the highest value is 850000**
- **The average value of serum_creatinine is 1.39 , the highest value is 9.4**
- **The average value of serum_sodium is 136.62 , the highest value is 148**
- **The average value of time is 130.26 , the highest value is 285**

4.4 | Check null Values

In [6]:

```
df.isnull().any() #to check "null" values
```

Out[6]:

```
age                         False
anaemia                     False
creatinine_phosphokinase    False
diabetes                    False
ejection_fraction           False
high_blood_pressure         False
platelets                   False
serum_creatinine            False
serum_sodium                False
sex                         False
smoking                     False
time                        False
DEATH_EVENT                 False
dtype: bool
```

5 DATA VISUALIZATION

In [15]:

```
plt.figure(figsize = (18, 10), dpi = 50)

sns.barplot(x = "anaemia", y = "creatinine_phosphokinase", hue = "sex",
            data = df_eda, palette = "Set1", saturation = 1);
```

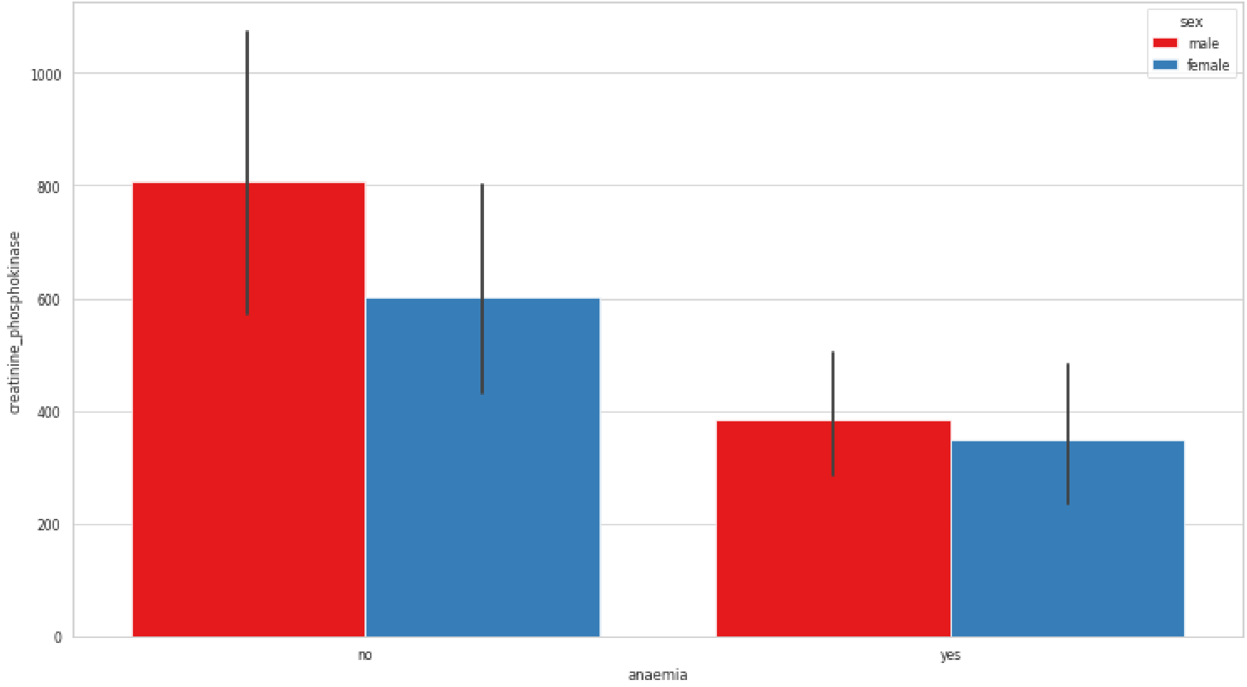


Chart report

From the graphs above, we can observe the relationships between various categorical and numerical variables.

5.2 Distplot

In [16]:

```python
fig, axes = plt.subplots(1, 2, figsize=(20, 7))

sns.distplot(ax = axes[0], x = df_eda["age"],
             hist = True,
             bins = 20,
             kde = True,
             vertical = False, color = "#C44D04").set(title = "The distribution of the values of 'age' variable");

sns.distplot(ax = axes[1], x = df_eda["platelets"],
             hist = True,
             bins = 20,
             kde = True,
             vertical = False, color = "#FA8B47").set(title = "The distribution of the values of 'platelets' variable");
```

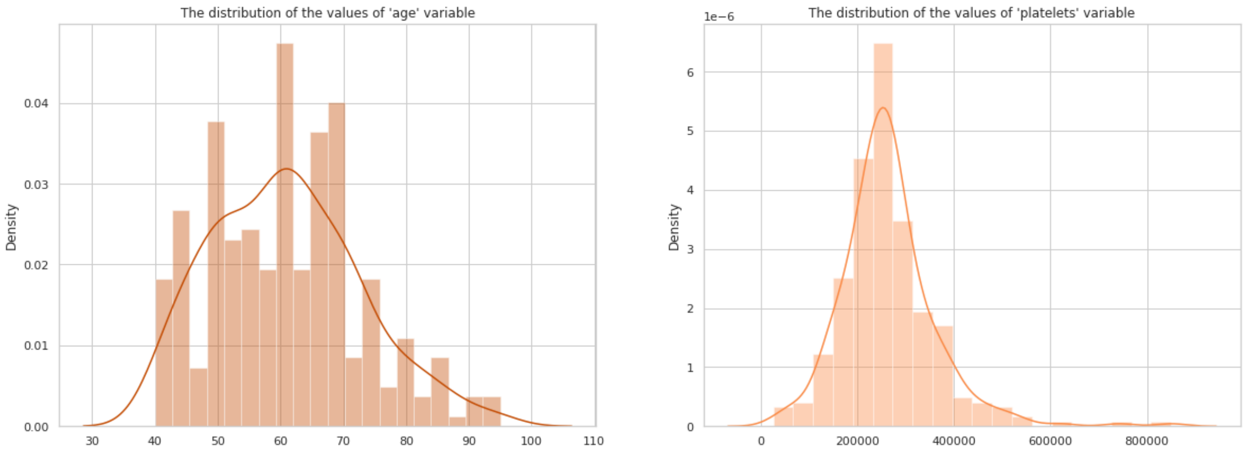


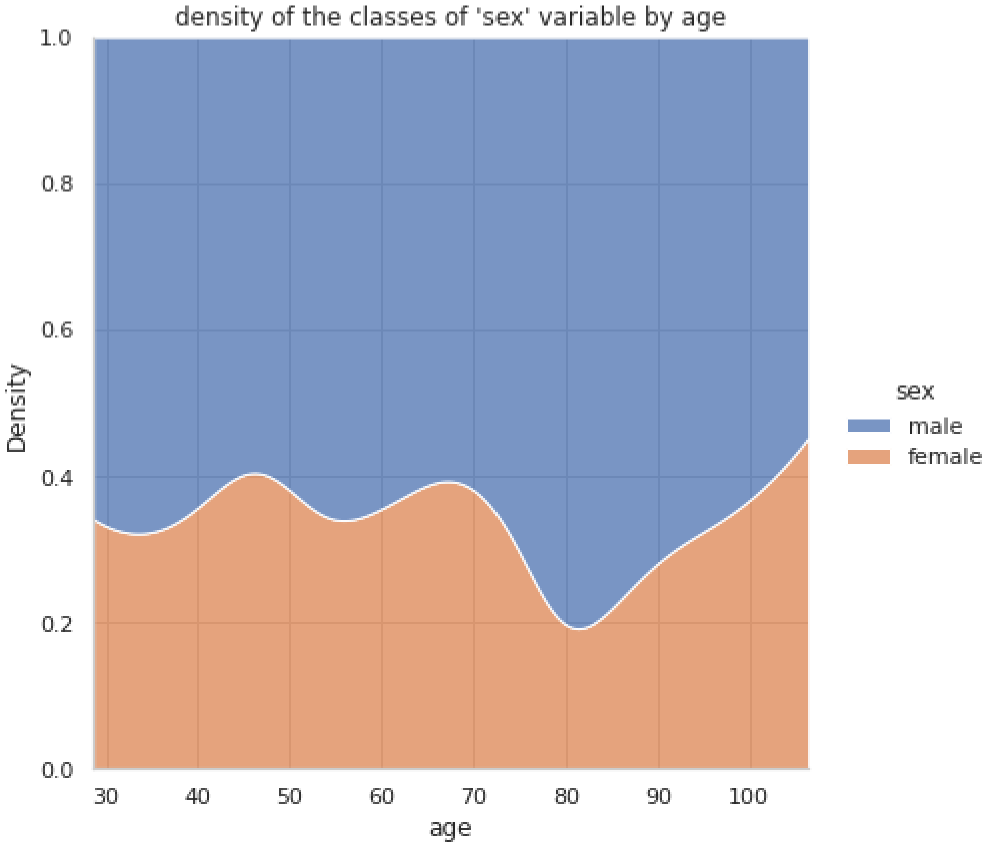
Chart report

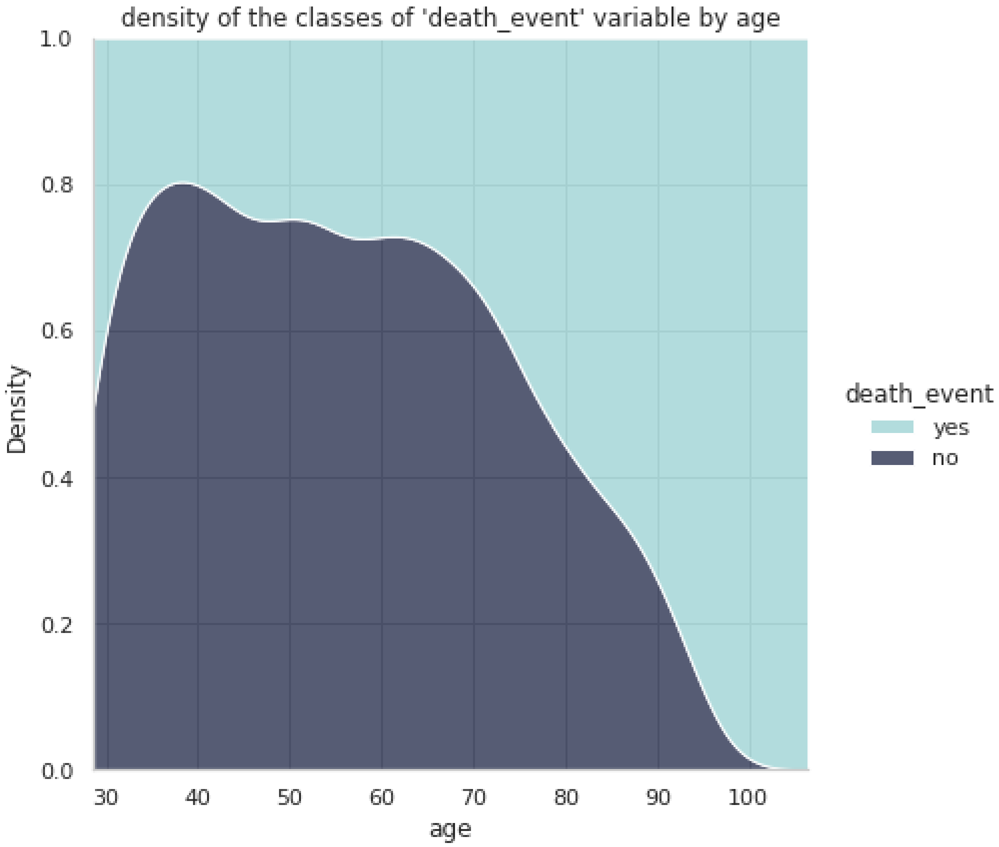
From the first graph, we can observe that the values of **the "age" variable** are mostly distributed as values **of 50, 60 and 65 - 70.** In addition, there are many **peaks** in the distribution graph of **the "age" variable.** This is an indicator that the values of the variable **are not normally distributed.** Looking at the second graph, we can observe that the values of **the "platelets" variable** are mainly distributed between **200000 and 400000.** At the same time, the graph **does not have many peaks, skewness and kurtosis are low.** It can be said that the values of this variable **are normally distributed.**

In [17]:

```
sns.displot(
    data = df_eda,
    x = "age", hue = "sex",
    kind="kde", height=6,
    multiple="fill", clip=(0, None),
    color = "#FE6203",
).set(title = "density of the classes of 'sex' variable by age");


sns.displot(
    data = df_eda,
    x = "age", hue = "death_event",
    kind = "kde", height=6,
    multiple="fill", clip=(0, None),
    palette="ch:rot=-.25,hue=1,light=.75",
).set(title = "density of the classes of 'death_event' variable by age");
```


density of the classes of 'sex' variable by age


density of the classes of 'death_event' variable by age

5.3 |Boxplot

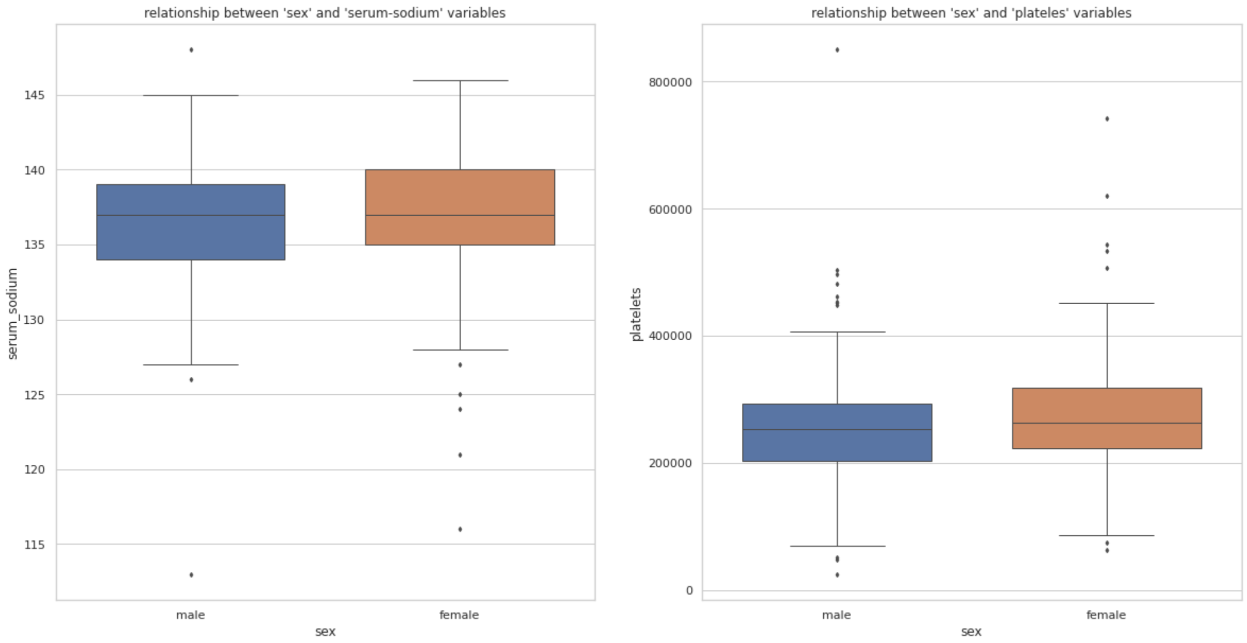
In [18]:

```
fig, axes = plt.subplots(1, 2, figsize=(20, 10))



sns.boxplot(ax = axes[0], x = "sex", y = "serum_sodium", data = df_eda, width = 0.7,
               orient = "v", fliersize = 3, linewidth = 1);
axes[0].set_title("relationship between 'sex' and 'serum-sodium' variables");

sns.boxplot(ax = axes[1], x = "sex", y = "platelets", data = df_eda, width = 0.7,
               orient = "v", fliersize = 3, linewidth = 1);
axes[1].set_title("relationship between 'sex' and 'plateles' variables");
```

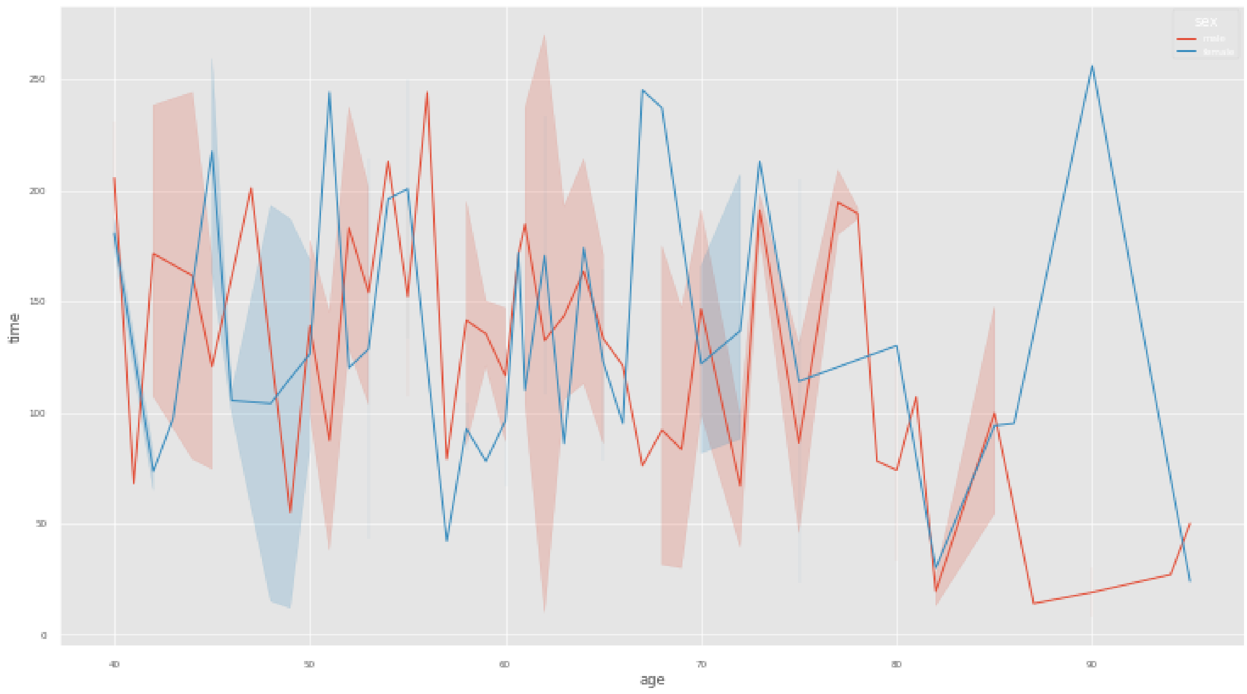


GG.plot

lo lo

In [33]:

```
plt.figure(figsize = (18, 10), dpi = 50)
plt.style.use('ggplot')
sns.lineplot(data = df_eda, x = "age", y = "time", hue = "sex" );
```



6.1 |Look at Dataset

In [34]:

```
df.head().style.background_gradient(cmap = "Reds_r")
```

Out[34]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_press
0	75.000000	0	582	0	20	1
1	55.000000	0	7861	0	38	0
2	65.000000	0	146	0	20	0
3	50.000000	1	111	0	20	0
4	65.000000	1	160	1	20	0

```
                    \n
```

```
DEPENDENT AND INDEPENDENT VARIABLES WERE SUCCESFULLY SELECTED...
```

6.3 |Split Dataset into Train and Test Sets

In [36]:

```
x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                    test_size = 0.20,
                                                    shuffle = True,
                                                    random_state = 1)

print(colored("\n THE DATASET WERE SUCCESFULLY SPLITTED - test = 20%, tra
in = 80%...", color = "green", attrs = ["bold", "dark"]))
```

```

 THE DATASET WERE SUCCESFULLY SPLITTED - test = 20%, train = 80%...
```

6.4 Standardization

In [37]:

```
scaler = StandardScaler()
scaler.fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)

print(colored("\n x_train AND x_test SETS WERE SUCCESFULLY STANDARIZED",
color = "green", attrs = ["bold", "dark"]))
```

```

 x_train AND x_test SETS WERE SUCCESFULLY STANDARIZED
```

In [38]:

```
x_train[0:5]
```

Out[38]:

```
array([[-1.33194278,  1.11069566, -0.4576634 , -0.84818893, -0.2431805
8,
        -0.72269841, -0.89319445, -0.58084862,  0.51465589,  0.7294184
5,
         1.52297224, -0.10575054],
       [-0.4999137 , -0.90033664, -0.51894319, -0.84818893,  0.1773704
9,
        -0.72269841, -0.57954879, -0.39670905,  0.29500811,  0.7294184
5,
        -0.65661079,  1.33189826],
       [ 0.58172409,  1.11069566,  0.39546617,  1.17898261, -0.2431805
8,
        -0.72269841,  0.15589621, -0.30463927, -0.58358301,  0.7294184
5,
        -0.65661079,  0.86979686],
       [-0.08389917,  1.11069566, -0.02487488, -0.84818893, -0.6637316
5,
         1.38370306, -1.40151673, -0.48877884,  1.83254257, -1.3709551
7,
        -0.65661079, -0.43949044],
       [-0.08389917, -0.90033664, -0.35712624,  1.17898261,  0.0091500
7,
        -0.72269841,  0.78318753,  1.44468663,  1.17359923, -1.3709551
7,
        -0.65661079, -1.27384019]])
```

In [39]:

```
x_test[0:5]
```

Out[39]:

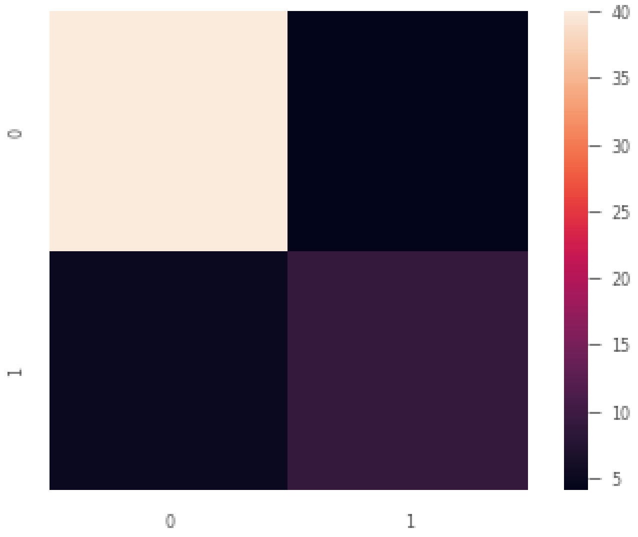
```
array([[-0.91592824,  1.11069566, -0.47202585, -0.84818893, -1.5048338
,
        -0.72269841, -0.73096394, -0.58084862,  0.51465589,  0.7294184
5,
        -0.65661079,  0.21515321],
       [-1.33194278, -0.90033664, -0.02487488,  1.17898261,  1.4390237
1,
        -0.72269841,  3.09767621, -0.39670905, -1.02287857, -1.3709551
7,
        -0.65661079,  1.55011281],
       [-0.66631952,  1.11069566, -0.49500577, -0.84818893, -1.5048338
,
         1.38370306,  1.74575525, -0.02842992,  0.51465589, -1.3709551
7,
        -0.65661079, -1.10697024],
       [-0.74952242, -0.90033664, -0.4557484 , -0.84818893, -0.6637316
5,
        -0.72269841, -0.41731828, -0.67291841, -0.14428745,  0.7294184
5,
         1.52297224, -0.22127589],
       [ 0.33211537,  1.11069566, -0.45287591, -0.84818893, -0.2431805
8,
         1.38370306,  0.36138819, -0.58084862, -0.58358301,  0.7294184
5,
        -0.65661079,  0.83128841]])
```

In [48]:

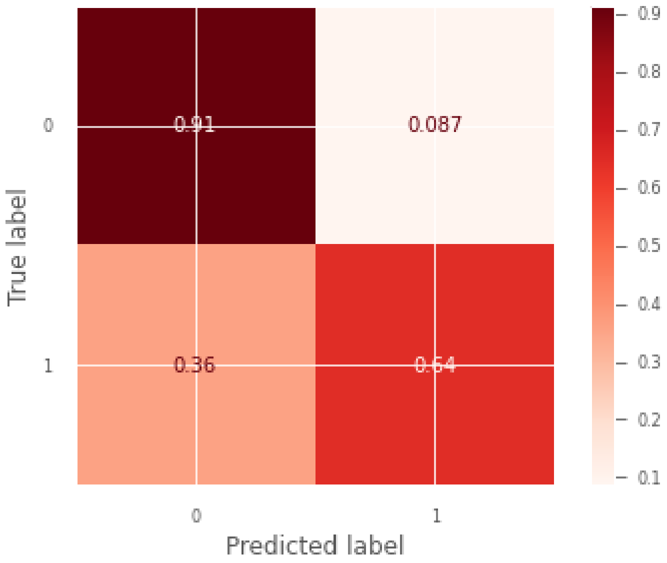
```
# Visualization Confusion Matrix
conf_mat = confusion_matrix(y_test, y_pred)
print(conf_mat)

# Visualize it as a heatmap
import seaborn
seaborn.heatmap(conf_mat, square = True, robust = True)
plt.show()
```

```
[[42  4]
 [ 5  9]]
```



```
In [49]:
plot_confusion_matrix(rf_model,
                      x_test,
                      y_test,
                      cmap = plt.cm.Reds,
                      normalize='true');
```



7.10 Logistic Regression

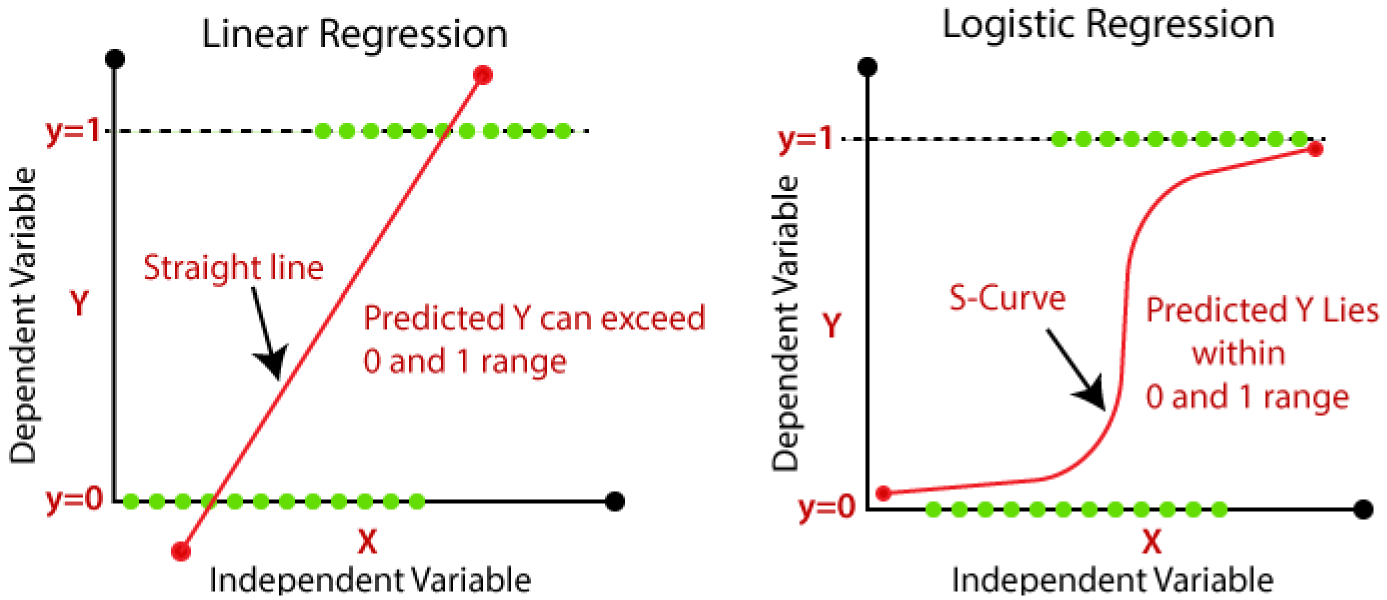
What is logistic regression?

This type of statistical model (also known as logit model) is often used for classification and predictive analytics. Logistic regression estimates the probability of an event occurring, such as voted or didn’t vote, based on a given dataset of independent variables. Since the outcome is a probability, **the dependent variable is bounded between 0 and 1.** In logistic regression, a logit transformation is applied on the odds - that is, the probability of success divided by the probability of failure. This is also commonly known as the log odds, or the natural logarithm of odds, and this logistic function is represented by the following formulas:

$$\text{Logit}(pi) = 1/(1+ \exp(-pi))$$

$$\ln(pi/(1-pi)) = Beta\_0 + Beta\_1*X\_1 + \ldots + B\_k*K\_k$$

In this logistic regression equation, **logit(pi) is the dependent or response variable and x is the independent variable.** The beta parameter, or coefficient, in this model is commonly estimated via maximum likelihood estimation (MLE). This method tests different values of beta through multiple iterations to optimize for the best fit of log odds. All of these iterations produce the log likelihood function, and logistic regression seeks to maximize this function to find the best parameter estimate. **Once the optimal coefficient (or coefficients if there is more than one independent variable) is found, the conditional probabilities for each observation can be calculated, logged, and summed together to yield a predicted probability.** For binary classification, a probability less than .5 will predict 0 while a probability **greater than 0 will predict 1.** After the model has been computed, it’s best practice to evaluate the how well the model predicts the dependent variable, which is called goodness of fit. The Hosmer–Lemeshow test is a popular method to assess model fit.



In [50]:

```
# build a model

log = LogisticRegression(solver = "liblinear")
log_model = log.fit(x_train, y_train)
log_model
```

Out[50]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept
=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001,
verbose=0,
                   warm_start=False)
```

7.11 Accuracy Score of Model on Test Set

In [51]:

```
y_pred = log_model.predict(x_test)
accuracy_score(y_test, y_pred)
```

Out[51]:

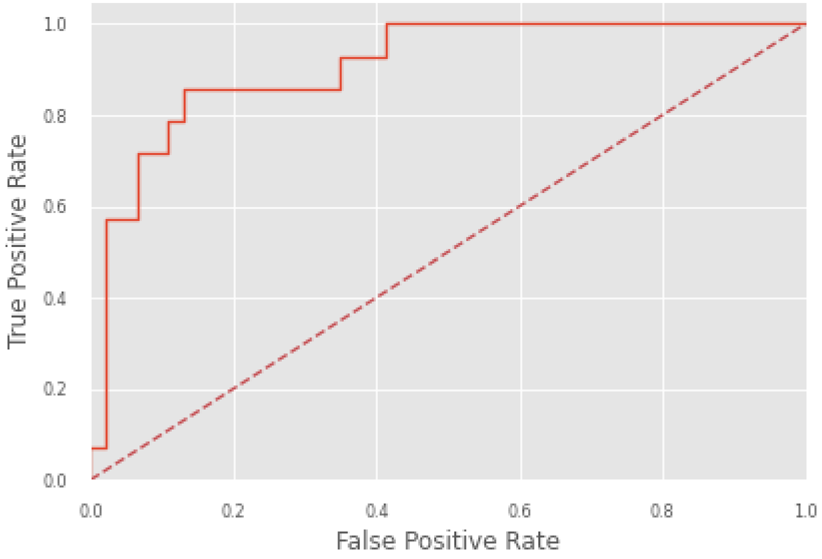
```
0.8833333333333333
```

7.12 | ROC AUC - Logistic Regression

In [52]:

```
log_roc_auc = roc_auc_score(y_test, log_model.predict(x_test))

fpr, tpr, thresholds = roc_curve(y_test, log_model.predict_proba(x_test)
[:,1])
plt.figure()
plt.plot(fpr, tpr, label = 'AUC (area = %0.2f)' % log_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
plt.show()
```


True Positive Rate
False Positive Rate