FINAL PROJECT

- **SOUPTIK DASGUPTA**

BACKGROUND

Heart failure is the state in which muscles in the heart wall get fade and enlarge, limiting heart pumping of blood. The ventricles of heart can get inflexible and do not fill properly between beats. With the passage of time heart fails in fulfilling the proper demand of blood in body and as a consequence person starts feeling difficulty in breathing.

The main reason behind heart failure include coronary heart disease, diabetes, high blood pressure and other diseases like HIV, alcohol abuse or cocaine, thyroid disorders, excess of vitamin E in body, radiation or chemotherapy, etc. As stated by WHO, Cardiovascular Heart Disease (CHD) is now top reason causing 31% of deaths globally. India is also included in the list of countries where prevalence of CHD is increasing significantly.

In addition to relative scarcity of studies focusing on heart failure, the present study has specific importance in the Indian context, as diet patterns in India are different with other the countries.

The main objective of this study is to estimate death rates due to heart failure and to investigate its link with some major risk factors.

DESIGN

The project is designed in a way where I have tried to use multiple classification models to predict heart failure as accurately as possible. The dataset has 13 columns which are as follows:

- Age (Float, Discrete)
- Anaemia (Binary)
- Creatinine_phosphokinase (Continous)
- Diabetes (Binary)
- Ejection_fraction (Discrete)
- High_blood_pressure (Binary)
- Platelets (Float, Discrete)
- Serum_creatinine (Continous)
- Seum_sodium (Discrete)
- Sex (Binary)
- Smoking (Binary)
- Time (Discrete)
- Death_Event (Binary)

We can use the first 12 columns as Independent Variables (IV) and the last column, Death_Event, as the Dependent Variable (DV) as we are trying to estimate that feature. As Death_Event, which is out Dependent Variable, is a Binary column, this case is a **Binary Classification** problem.

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [2]:

```
df = pd.read_csv("C:/Users/LENOVO/Downloads/Project IITR/heart_failure_dataset.csv")
df.sample(5)
```

Out[2]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sod
100	65.0	1	305	0	25	0	298000.0	1.1	
93	60.0	1	154	0	25	0	210000.0	1.7	
32	50.0	1	249	1	35	1	319000.0	1.0	
289	90.0	1	337	0	38	0	390000.0	0.9	
138	62.0	0	281	1	35	0	221000.0	1.0	

In [3]:

```
df.describe()
```

Out[3]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_cre
count	299.000000	299.000000	299.000000	299.000000	299.000000	299.000000	299.000000	29
mean	60.833893	0.431438	581.839465	0.418060	38.083612	0.351171	263358.029264	
std	11.894809	0.496107	970.287881	0.494067	11.834841	0.478136	97804.236869	
min	40.000000	0.000000	23.000000	0.000000	14.000000	0.000000	25100.000000	
25%	51.000000	0.000000	116.500000	0.000000	30.000000	0.000000	212500.000000	
50%	60.000000	0.000000	250.000000	0.000000	38.000000	0.000000	262000.000000	
75%	70.000000	1.000000	582.000000	1.000000	45.000000	1.000000	303500.000000	
max	95.000000	1.000000	7861.000000	1.000000	80.000000	1.000000	850000.000000	9

In [4]:

```
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 [5]:

```
df.iloc[:,2]
```

Out[5]:

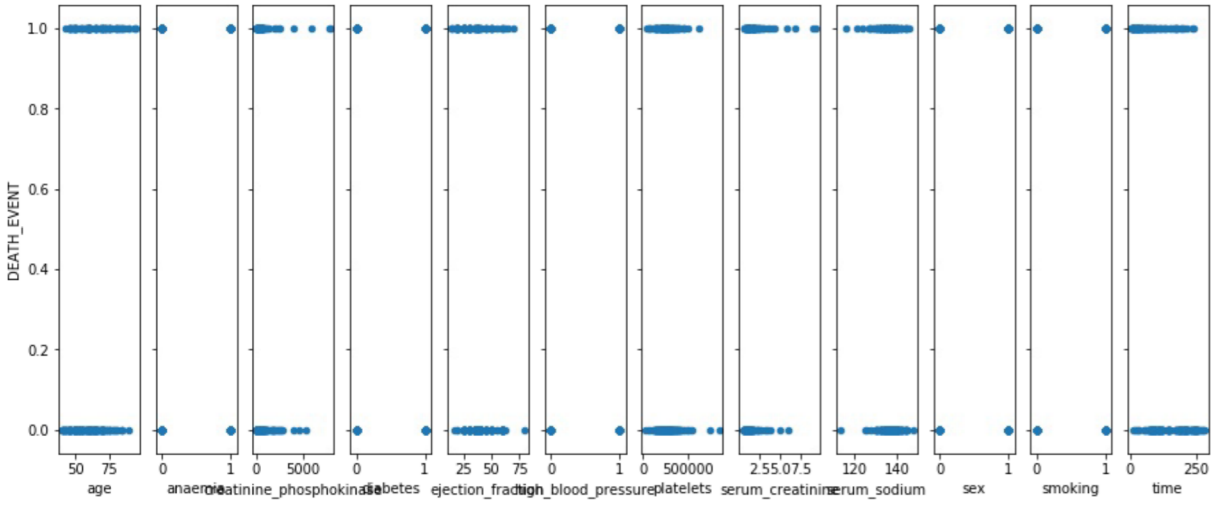
```
0       582
1      7861
2       146
3       111
4       160
       ...
294      61
295    1820
296    2060
297    2413
298     196
Name: creatinine_phosphokinase, Length: 299, dtype: int64
```

In [6]:

```
fig, axs = plt.subplots(1,12, sharey = True)
df.plot(x = df.columns[0], y = df.columns[12], kind = "scatter", ax = axs[0], figsize = (15,6))
df.plot(x = df.columns[1], y = df.columns[12], kind = "scatter", ax = axs[1], figsize = (15,6))
df.plot(x = df.columns[2], y = df.columns[12], kind = "scatter", ax = axs[2], figsize = (15,6))
df.plot(x = df.columns[3], y = df.columns[12], kind = "scatter", ax = axs[3], figsize = (15,6))
df.plot(x = df.columns[4], y = df.columns[12], kind = "scatter", ax = axs[4], figsize = (15,6))
df.plot(x = df.columns[5], y = df.columns[12], kind = "scatter", ax = axs[5], figsize = (15,6))
df.plot(x = df.columns[6], y = df.columns[12], kind = "scatter", ax = axs[6], figsize = (15,6))
df.plot(x = df.columns[7], y = df.columns[12], kind = "scatter", ax = axs[7], figsize = (15,6))
df.plot(x = df.columns[8], y = df.columns[12], kind = "scatter", ax = axs[8], figsize = (15,6))
df.plot(x = df.columns[9], y = df.columns[12], kind = "scatter", ax = axs[9], figsize = (15,6))
df.plot(x = df.columns[10], y = df.columns[12], kind = "scatter", ax = axs[10], figsize = (15,6))
df.plot(x = df.columns[11], y = df.columns[12], kind = "scatter", ax = axs[11], figsize = (15,6))
```

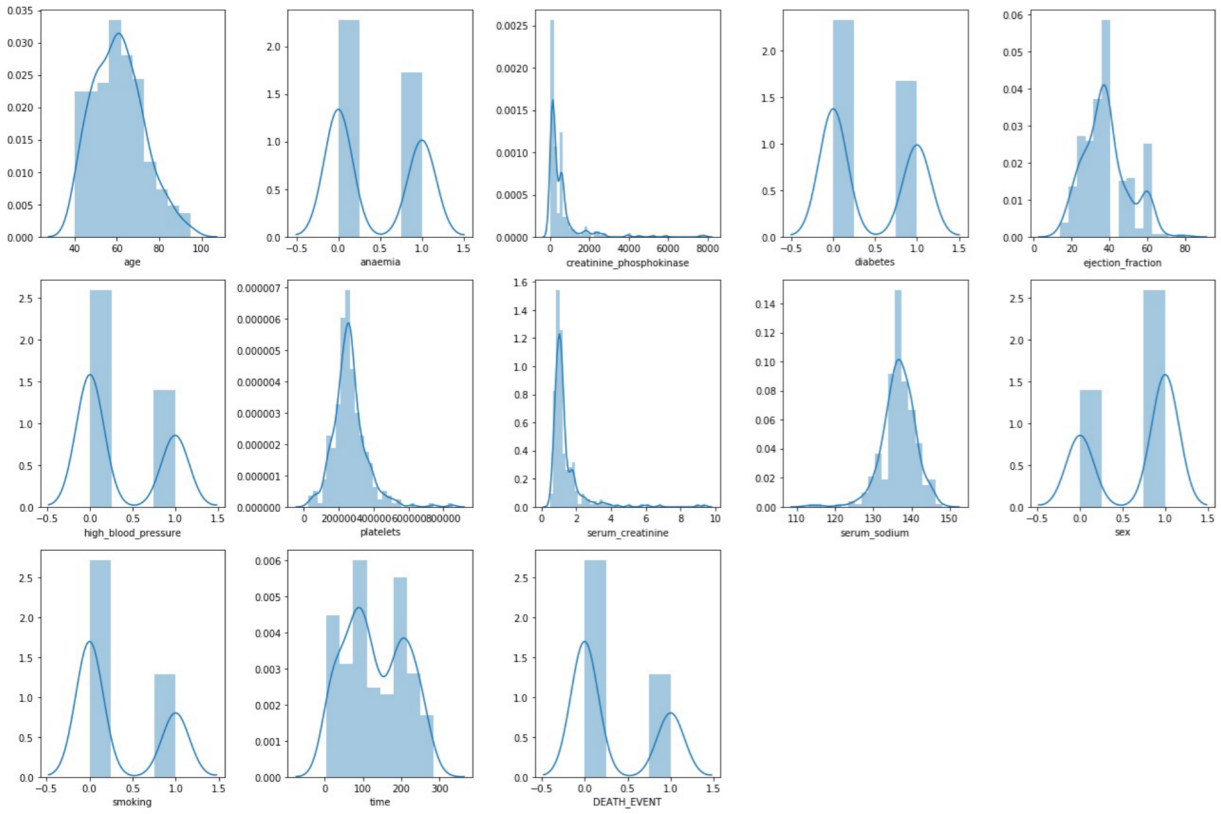
Out[6]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x238544d2988>
```



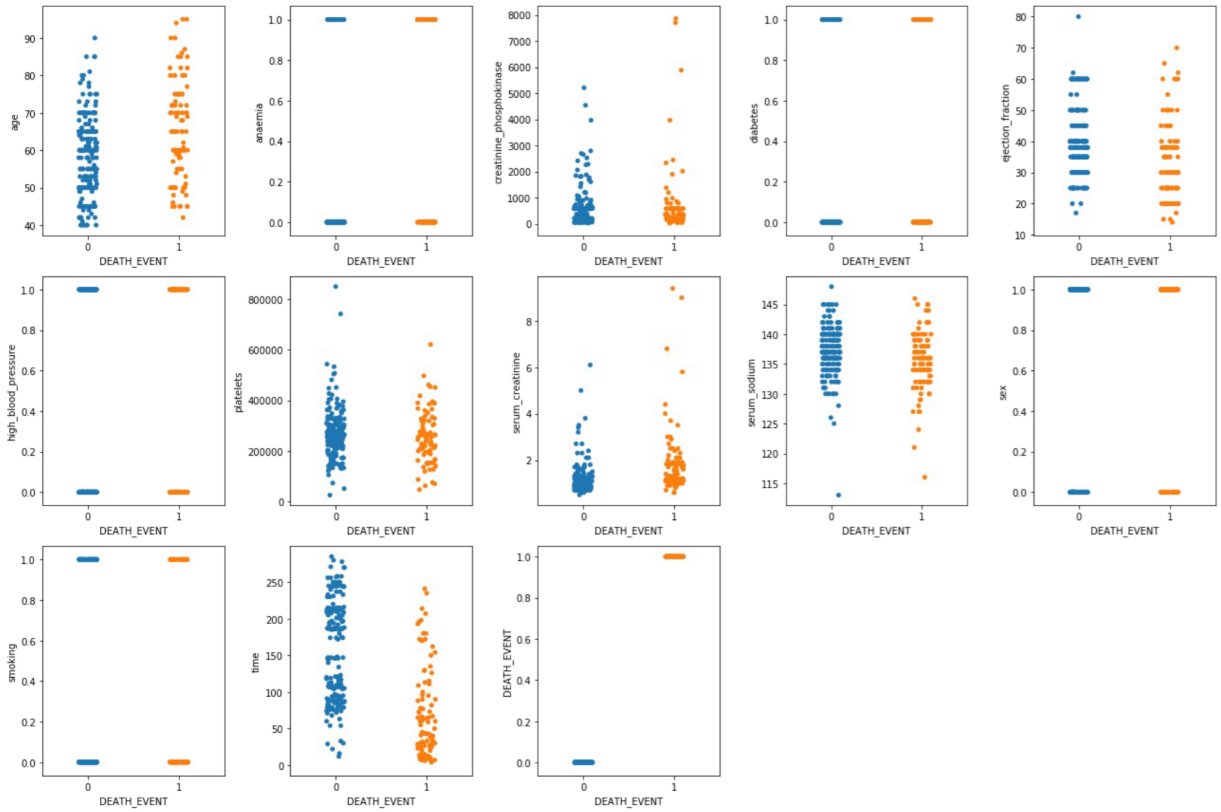
In [7]:

```
plt.figure(figsize = (18,12), facecolor = "white")

i = 1

for col in df:
    if i < 14:
        plt.subplot(3,5,i)
        sns.distplot(df[col])
        xlabel = col
    i += 1


plt.tight_layout()
```



In [8]:

```
plt.figure(figsize = (18,12))

i = 1

for col in df:
    if i < 14:
        plt.subplot(3,5,i)
        sns.stripplot(df["DEATH_EVENT"], df[col])
        xlabel = col
    i += 1


plt.tight_layout()
```

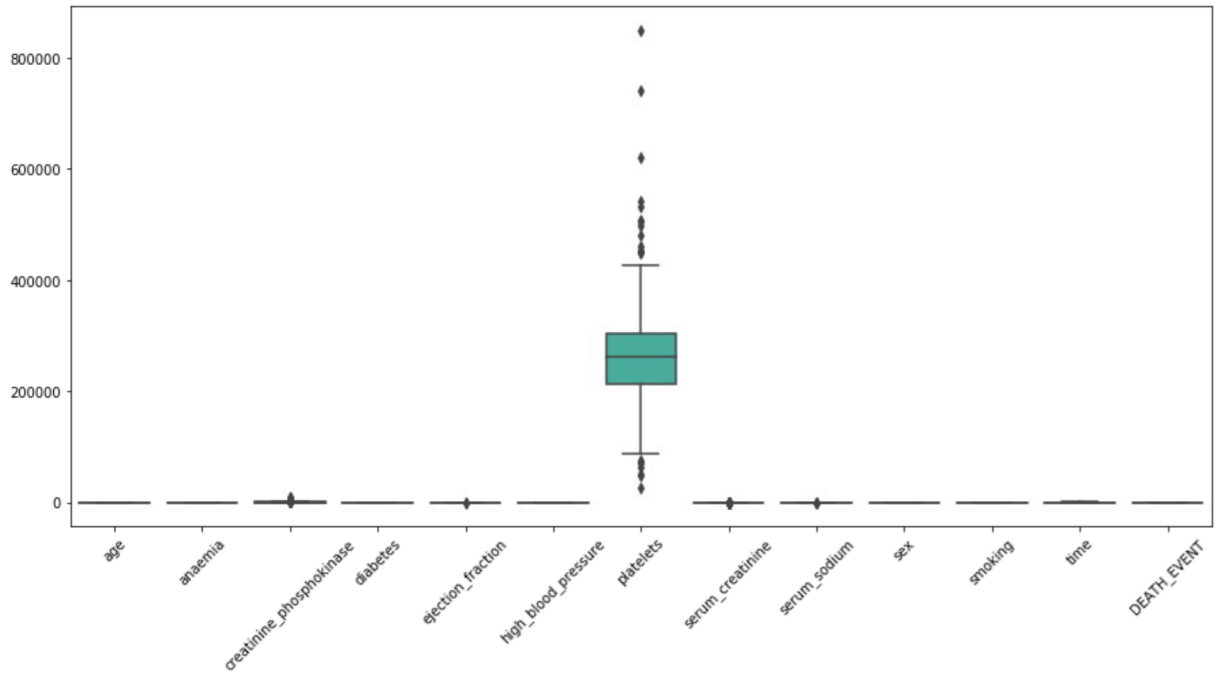


In [9]:

```
fig, axs = plt.subplots(figsize = (15,7), sharey = True)
sns.boxplot(data = df)
plt.xticks(rotation = 45)
```

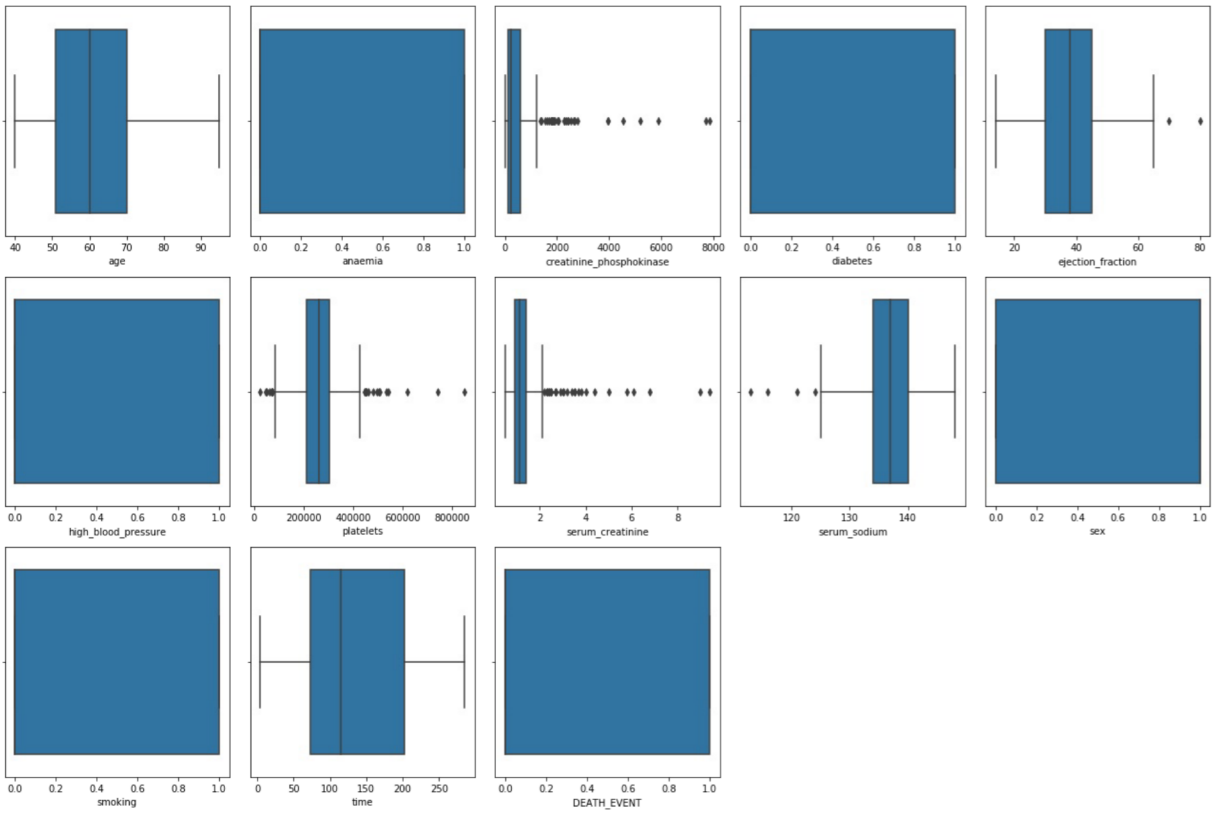
Out[9]:

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12]),
 <a list of 13 Text xticklabel objects>)
```



In [10]:

```
plt.figure(figsize = (18,12))

i = 1

for col in df:
    if i < 14:
        plt.subplot(3,5,i)
        sns.boxplot(df[col])
        xlabel = col
    i += 1


plt.tight_layout()
```



In [11]:

```
df.columns
```

Out[11]:

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

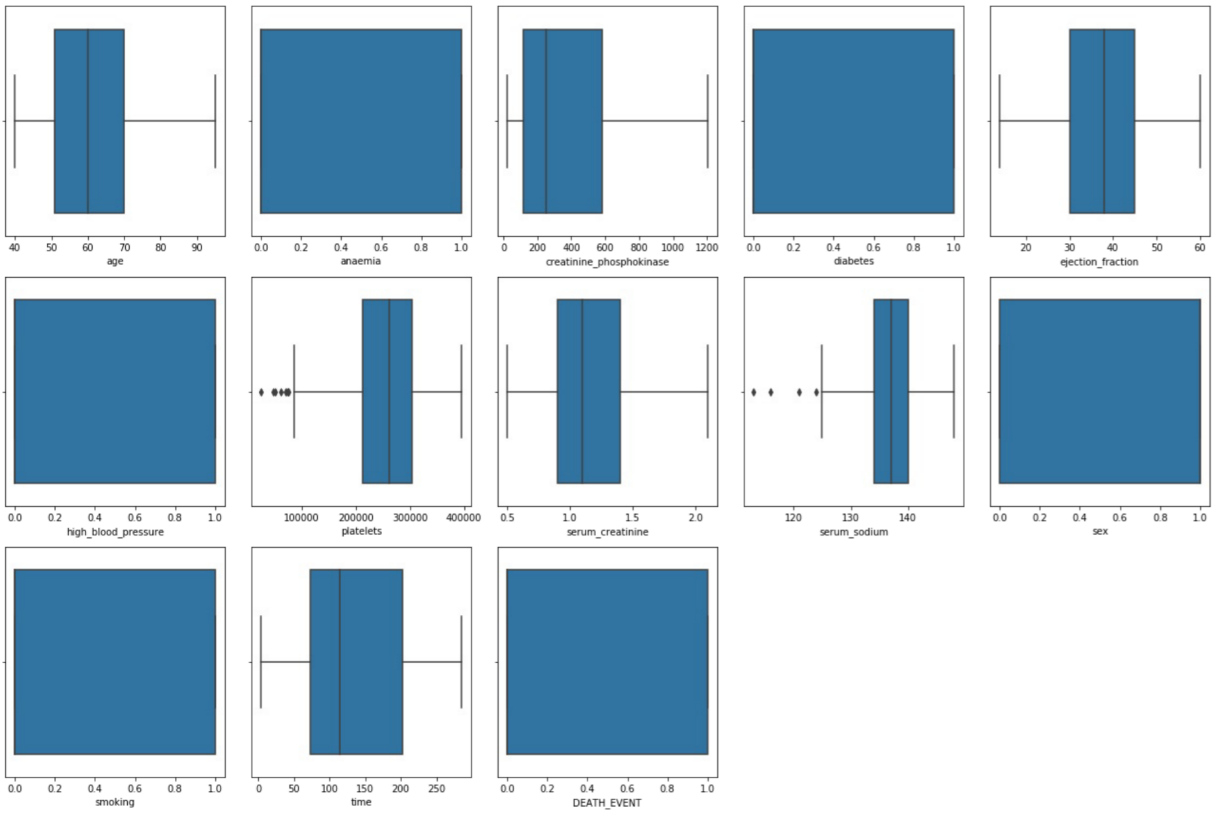
In [12]:

```
df["creatinine_phosphokinase"][df["creatinine_phosphokinase"] > df["creatinine_phosphokinase"].quantile(0.9)] = d
f["creatinine_phosphokinase"].quantile(0.9)
df["ejection_fraction"][df["ejection_fraction"] > df["ejection_fraction"].quantile(0.98)] = df["ejection_fraction
"].quantile(0.98)
df["platelets"][df["platelets"] > df["platelets"].quantile(0.93)] = df["platelets"].quantile(0.93)
df["serum_creatinine"][df["serum_creatinine"] > df["serum_creatinine"].quantile(0.9)] = df["serum_creatinine"].qu
antile(0.9)
```

```
C:\Users\LENOVO\anaconda3\lib\site-packages\ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexi
ng.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
C:\Users\LENOVO\anaconda3\lib\site-packages\ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexi
ng.html#returning-a-view-versus-a-copy

C:\Users\LENOVO\anaconda3\lib\site-packages\ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexi
ng.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports until
C:\Users\LENOVO\anaconda3\lib\site-packages\ipykernel_launcher.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexi
ng.html#returning-a-view-versus-a-copy
  after removing the cwd from sys.path.
```

In [13]:

```
plt.figure(figsize = (18,12))

i = 1

for col in df:
    if i < 14:
        plt.subplot(3,5,i)
        sns.boxplot(df[col])
        xlabel = col
    i += 1


plt.tight_layout()
```



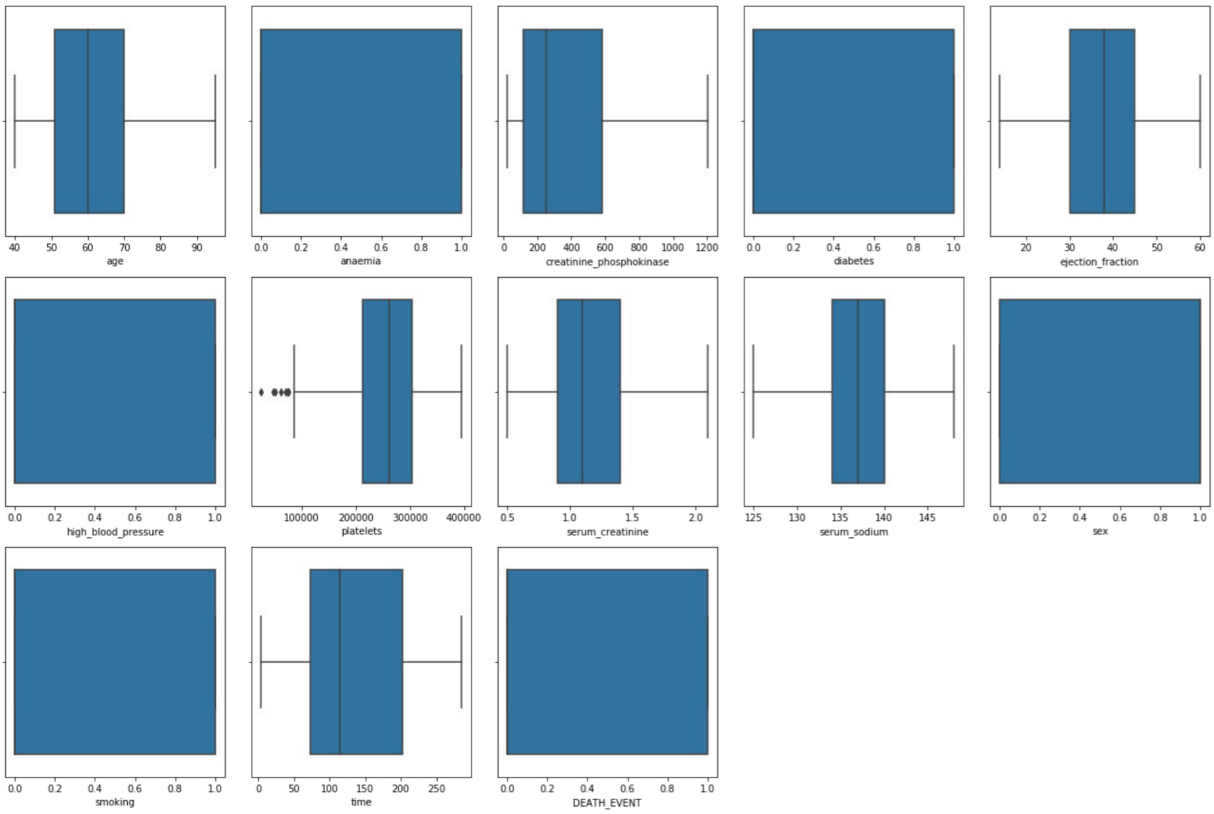
In [14]:

```
df["serum_sodium"][df["serum_sodium"] < 125] = 125
df["serum_sodium"][df["serum_sodium"] < 125] = 125
```

```
C:\Users\LENOVO\anaconda3\lib\site-packages\ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexi
ng.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
C:\Users\LENOVO\anaconda3\lib\site-packages\ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexi
ng.html#returning-a-view-versus-a-copy
```

In [15]:

```
plt.figure(figsize = (18,12))

i = 1

for col in df:
    if i < 14:
        plt.subplot(3,5,i)
        sns.boxplot(df[col])
        xlabel = col
    i += 1


plt.tight_layout()
```



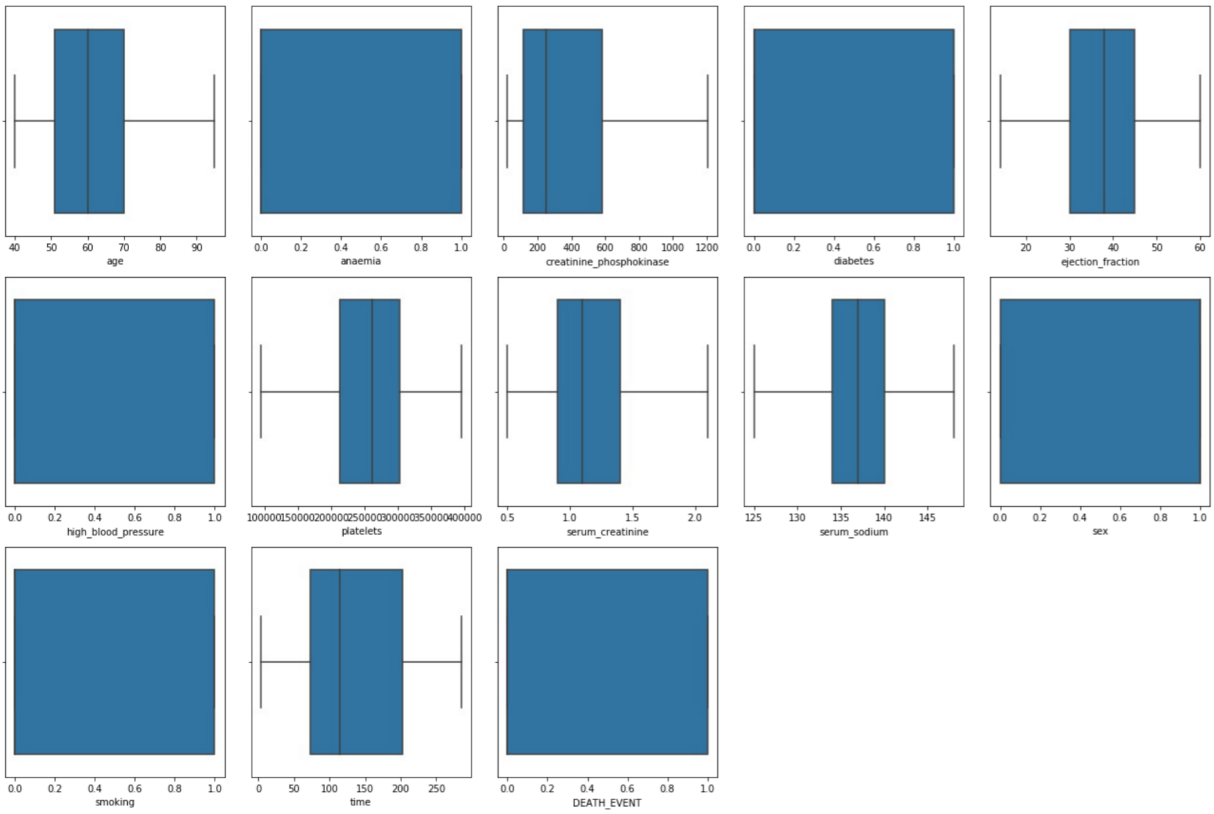
In [16]:

```
df["platelets"][df["platelets"] < df["platelets"].quantile(0.025)] = df["platelets"].quantile(0.025)
```

```
C:\Users\LENOVO\anaconda3\lib\site-packages\ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexi
ng.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
```

In [17]:

```
plt.figure(figsize = (18,12))

i = 1

for col in df:
    if i < 14:
        plt.subplot(3,5,i)
        sns.boxplot(df[col])
        xlabel = col
    i += 1


plt.tight_layout()
```



In [18]:

```
X = df.iloc[:,:-1]
y = df.iloc[:,-1]
```

In [19]:

```
from sklearn.preprocessing import scale
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, f1_score, roc_auc_score, pre
cision_score, recall_score, roc_curve, auc
from sklearn.linear_model import LogisticRegression
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import train_test_split, GridSearchCV
```

In [20]:

```
X = scale(X)
X
```

Out[20]:

```
array([[ 1.19294523, -0.87110478,  0.44826852, ...,  0.73568819,
        -0.68768191, -1.62950241],
       [-0.49127928, -0.87110478,  2.13527061, ...,  0.73568819,
        -0.68768191, -1.60369074],
       [ 0.35083298, -0.87110478, -0.73464063, ...,  0.73568819,
         1.4541607 , -1.5907849 ],
       ...,
       [-1.33339153, -0.87110478,  2.13527061, ..., -1.35927151,
        -0.68768191,  1.90669738],
       [-1.33339153, -0.87110478,  2.13527061, ...,  0.73568819,
         1.4541607 ,  1.93250906],
       [-0.9123354 , -0.87110478, -0.59898591, ...,  0.73568819,
         1.4541607 ,  1.99703825]])
```

In [21]:

```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2, random_state = 10)
```

In [22]:

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[22]:

```
((239, 12), (60, 12), (239,), (60,))
```

In [23]:

```
X.shape
```

Out[23]:

```
(299, 12)
```

Logistic Regression

In [24]:

```
VIF = pd.DataFrame()
VIF["Columns"] = df.iloc[:,:-1].columns
VIF["VIF"] = [variance_inflation_factor(X,i) for i in range(X.shape[1])]
```

In [25]:

```
#no multicolinearity
VIF
```

Out[25]:

	Columns	VIF
0	age	1.159810
1	anaemia	1.094779
2	creatinine_phosphokinase	1.076398
3	diabetes	1.054045
4	ejection_fraction	1.097856
5	high_blood_pressure	1.082719
6	platelets	1.037056
7	serum_creatinine	1.244371
8	serum_sodium	1.129018
9	sex	1.326540
10	smoking	1.288867
11	time	1.142260

In [26]:

```
logistic regression – LogisticRegression()
```

In [27]:

```
logistic_regression.fit(X_train, y_train)
```

Out[27]:

```
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='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [28]:

```
pred = logistic_regression.predict(X_test)
pred
```

Out[28]:

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

In [29]:

```
#training accuracy
logistic_regression.score(X_train, y_train)
```

Out[29]:

```
0.8619246861924686
```

In [175]:

```
#testing accuracy
logistic_accuracy = accuracy_score(y_test, pred)
logistic_accuracy
```

Out[175]:

```
0.7666666666666667
```

In [176]:

```
logistic_precision = precision_score(y_test, pred)
logistic_precision
```

Out[176]:

```
0.5714285714285714
```

In [177]:

```
logistic_recall = recall_score(y_test, pred)
logistic_recall
```

Out[177]:

```
0.7058823529411765
```

In [178]:

```
logistic_f1 = f1_score(y_test, pred)
logistic_f1
```

Out[178]:

```
0.6315789473684211
```

In [34]:

```
roc = roc_auc_score(y_test, pred)
roc
```

Out[34]:

```
0.7482900136798906
```

In [35]:

```
print(classification_report(y_test, pred))
```

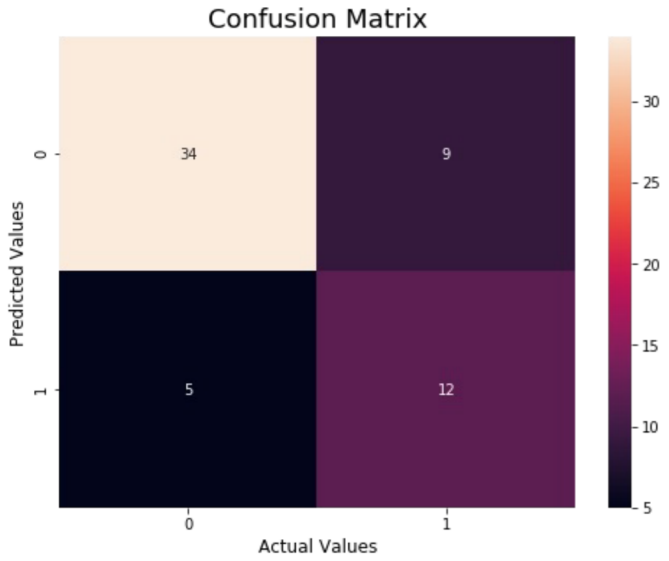
```
              precision    recall  f1-score   support

           0       0.87      0.79      0.83        43
           1       0.57      0.71      0.63        17

    accuracy                           0.77        60
   macro avg       0.72      0.75      0.73        60
weighted avg       0.79      0.77      0.77        60
```

In [36]:

```
plt.figure(figsize = (8,6))
sns.heatmap(confusion_matrix(y_test, pred), annot = True)
plt.xlabel("Actual Values", fontsize = 12)
plt.ylabel("Predicted Values", fontsize = 12)
plt.title("Confusion Matrix", fontsize = 18)
```

Out[36]:

```
Text(0.5, 1, 'Confusion Matrix')
```

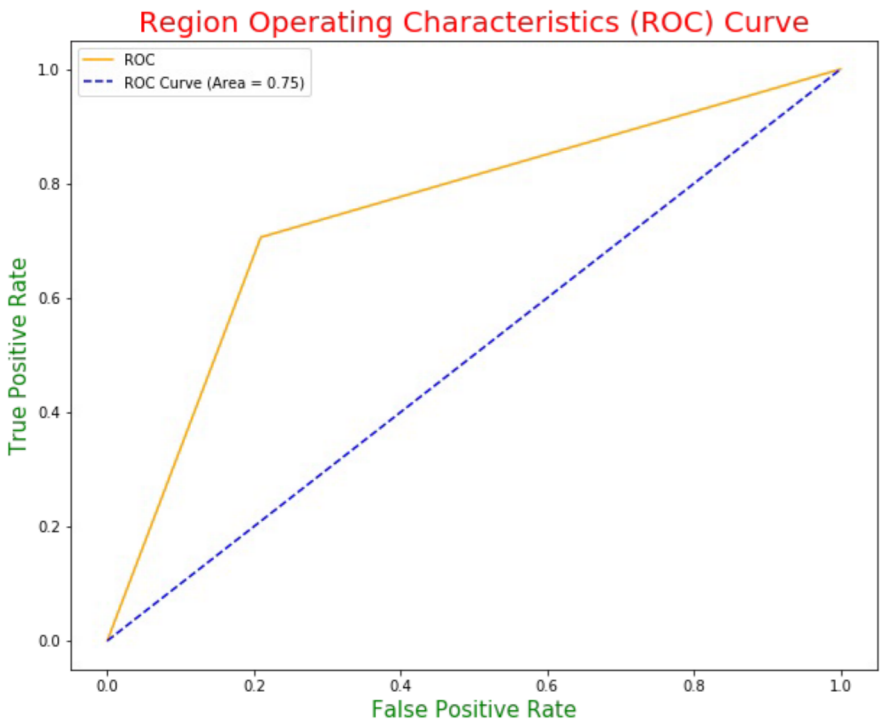


In [37]:

```
fpr, tpr, threshold = roc_curve(y_test, pred)
```

In [38]:

```
plt.figure(figsize = (10,8))
plt.plot(fpr, tpr, color = 'orange', label = "ROC")
plt.plot([0,1], [0,1], color = 'blue', linestyle = "--", label = "ROC Curve (Area = %0.2f)" %roc)
plt.legend()
plt.xlabel("False Positive Rate", fontsize = 15, color = "green")
plt.ylabel("True Positive Rate", fontsize = 15, color = "green")
plt.title("Region Operating Characteristics (ROC) Curve", fontsize = 20, color = "red")
plt.show()
```



K-Nearest Neighbour

In [39]:

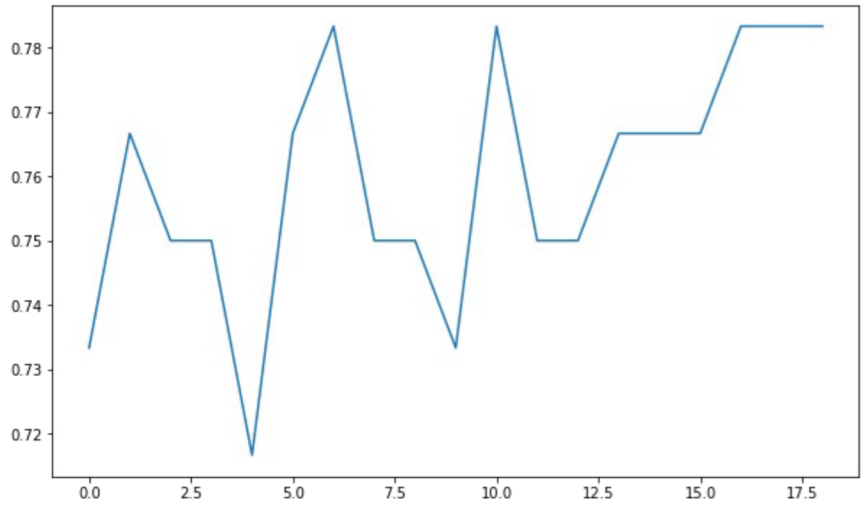
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import BaggingClassifier
```

In [40]:

```
scores = []

for i in range(1,20):

    knn = KNeighborsClassifier(n_neighbors = i)
    knn.fit(X_train, y_train)
    scores.append(knn.score(X_test, y_test))

plt.figure(figsize = (10,6))
plt.plot(scores)
```

Out[40]:

```
[<matplotlib.lines.Line2D at 0x23854ae4c88>]
```



In [41]:

```
knn = KNeighborsClassifier(n_neighbors = 6)
```

In [42]:

```
knn.fit(X_train, y_train)
```

Out[42]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=6, p=2,
                     weights='uniform')
```

In [43]:

```
pred_knn = knn.predict(X_test)
pred_knn
```

Out[43]:

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

In [44]:

```
#training accuracy
knn.score(X train, y train)
```

Out[44]:

```
0.799163179916318
```

In [179]:

```
#testing accuracy
knn_accuracy = accuracy_score(y_test, pred_knn)
knn_accuracy
```

Out[179]:

```
0.7666666666666667
```

In [46]:

```
knn_bagging = BaggingClassifier(

    base_estimator=knn,
    n_estimators = 100,
    max_samples = 0.5,
    bootstrap = True,
    n_jobs = -1,
    verbose = 2)
```

In [47]:

```
knn_bagging.fit(X_train, y_train)
```

```
[Parallel(n_jobs=8)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done   3 out of   8 | elapsed:    1.5s remaining:    2.5s
[Parallel(n_jobs=8)]: Done   8 out of   8 | elapsed:    1.6s remaining:    0.0s
[Parallel(n_jobs=8)]: Done   8 out of   8 | elapsed:    1.6s finished
```

Out[47]:

```
BaggingClassifier(base_estimator=KNeighborsClassifier(algorithm='auto',
                                                      leaf_size=30,
                                                      metric='minkowski',
                                                      metric_params=None,
                                                      n_jobs=None,
                                                      n_neighbors=6, p=2,
                                                      weights='uniform'),
                  bootstrap=True, bootstrap_features=False, max_features=1.0,
                  max_samples=0.5, n_estimators=100, n_jobs=-1, oob_score=False,
                  random_state=None, verbose=2, warm_start=False)
```

In [48]:

```
#training accuracy
knn_bagging.score(X_train, y_train)
```

```
[Parallel(n_jobs=8)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done   3 out of   8 | elapsed:    0.0s remaining:    0.1s
[Parallel(n_jobs=8)]: Done   8 out of   8 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=8)]: Done   8 out of   8 | elapsed:    0.0s finished
```

Out[48]:

```
0.7824267782426778
```

In [180]:

```
#testing accuracy
knn_accuracy = knn_bagging.score(X_test, y_test)
knn_accuracy
```

```
[Parallel(n_jobs=8)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done   3 out of   8 | elapsed:    1.9s remaining:    3.2s
[Parallel(n_jobs=8)]: Done   8 out of   8 | elapsed:    1.9s remaining:    0.0s
[Parallel(n_jobs=8)]: Done   8 out of   8 | elapsed:    1.9s finished
```

Out[180]:

```
0.75
```

In [50]:

```
knn_pred = knn_bagging.predict(X_test)
knn_pred
```

```
[Parallel(n_jobs=8)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done   3 out of   8 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=8)]: Done   8 out of   8 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=8)]: Done   8 out of   8 | elapsed:    0.0s finished
```

Out[50]:

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

In [181]:

```
knn_precision = precision_score(y_test, knn_pred)
knn_precision
```

Out[181]:

```
0.6
```

In [182]:

```
knn_recall = recall_score(y_test, knn_pred)
knn_recall
```

Out[182]:

```
0.35294117647058826
```

In [183]:

```
knn_f1 = f1_score(y_test, knn_pred)
knn_f1
```

Out[183]:

```
0.4444444444444445
```

Clustering

In [52]:

```
from sklearn.cluster import KMeans
```

In [53]:

```
#because binary classification
kmeans = KMeans(n_clusters = 2)
```

In [54]:

```
kmeans.fit(X_train, y_train)
```

Out[54]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=2, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

In [55]:

```
pred_kmeans = kmeans.predict(X_test)
pred_kmeans
```

Out[55]:

```
array([1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
       0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0,
       1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [184]:

```
kmeans_accuracy = accuracy_score(y_test, pred_kmeans)
kmeans_accuracy
```

Out[184]:

```
0.5
```

In [185]:

```
kmeans_precision = precision_score(y_test, pred_kmeans)
kmeans_precision
```

Out[185]:

```
0.24
```

In [186]:

```
kmeans_recall = recall_score(y_test, pred_kmeans)
kmeans_recall
```

Out[186]:

```
0.35294117647058826
```

In [187]:

```
kmeans_f1 = f1_score(y_test, pred_kmeans)
kmeans_f1
```

Out[187]:

```
0.28571428571428564
```

XGBoost

In [57]:

```
from xgboost import XGBClassifier
```

In [58]:

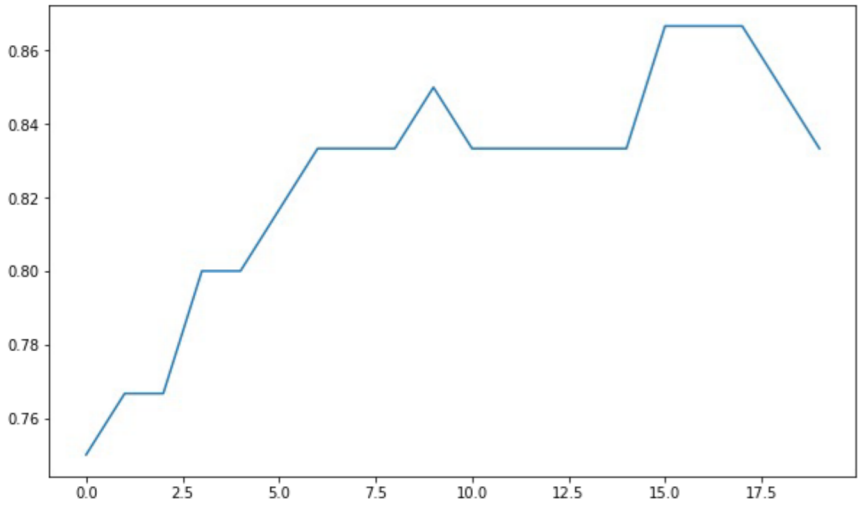
```
scores = []

for i in range(10,30):

    xgboost = XGBClassifier(n_estimators = i)
    xgboost.fit(X_train, y_train)
    scores.append(xgboost.score(X_test, y_test))

plt.figure(figsize = (10,6))
plt.plot(scores)
```

```
C:\Users\LENOVO\anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of label e
ncoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warnin
g, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object;
and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
```

```
[00:42:34] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115
: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic'
was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old b
ehavior.
[00:42:34] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115
: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic'
was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old b
ehavior.
[00:42:34] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115
: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic'
was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old b
ehavior.
[00:42:34] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115
: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic'
was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old b
ehavior.
[00:42:34] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115
: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic'
was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old b
ehavior.
[00:42:34] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115
: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic'
was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old b
ehavior.
[00:42:34] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115
: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic'
was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old b
ehavior.
[00:42:34] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115
: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic'
was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old b
ehavior.
[00:42:34] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115
: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic'
was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old b
ehavior.
[00:42:34] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115
: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic'
was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old b
ehavior.
[00:42:34] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115
: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic'
was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old b
ehavior.
[00:42:35] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115
: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic'
was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old b
ehavior.
[00:42:35] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115
: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic'
was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old b
ehavior.
[00:42:35] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115
: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic'
was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old b
ehavior.
[00:42:35] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115
: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic'
was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old b
ehavior.
[00:42:35] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115
: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic'
was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old b
ehavior.
[00:42:35] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115
: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic'
was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old b
ehavior.
[00:42:35] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115
: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic'
was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old b
ehavior.
[00:42:35] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115
: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic'
was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old b
ehavior.
[00:42:35] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115
: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic'
was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old b
ehavior.
```

Out[58]:

```
[<matplotlib.lines.Line2D at 0x2385880dec8>]
```



In [59]:

```
xgboost = XGBClassifier(n_estimators = 16)
```

In [60]:

```
xgboost.fit(X_train, y_train)
```

```
[00:42:35] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115
: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic'
was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old b
ehavior.

C:\Users\LENOVO\anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of label e
ncoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warnin
g, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object;
and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
```

Out[60]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
              gamma=0, gpu_id=-1, importance_type=None,
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=16, n_jobs=8,
              num_parallel_tree=1, objective='binary:logistic',
              predictor='auto', random_state=0, reg_alpha=0, reg_lambda=1,
              scale_pos_weight=1, subsample=1, tree_method='exact',
              use_label_encoder=True, validate_parameters=1, verbosity=None)
```

In [156]:

```
pred_xgboost = xgboost.predict(X_test)
pred_xgboost
```

Out[156]:

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

In [157]:

```
#training accuracy
xgboost.score(X_train, y_train)
```

Out[157]:

```
0.99581589958159
```

In [188]:

```
#testing accuracy
xgboost_accuracy = xgboost.score(X_test, y_test)
xgboost_accuracy
```

Out[188]:

```
0.8333333333333334
```

In [189]:

```
xgboost_precision = precision_score(y_test, pred_xgboost)
xgboost_precision
```

Out[189]:

```
0.7333333333333333
```

In [190]:

```
xgboost_recall = recall_score(y_test, pred_xgboost)
xgboost_recall
```

Out[190]:

```
0.6470588235294118
```

In [191]:

```
xgboost_f1 = f1_score(y_test, pred_xgboost)
xgboost_f1
```

Out[191]:

```
0.6875
```

Decision Tree

In [64]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

In [65]:

```
decision_tree = DecisionTreeClassifier(criterion = 'entropy')
```

In [66]:

```
decision_tree.fit(X_train, y_train)
```

Out[66]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

In [67]:

```
pred_decision_tree = decision_tree.predict(X_test)
pred_decision_tree
```

Out[67]:

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

In [68]:

```
#training accuracy
decision_tree.score(X_train, y_train)
```

Out[68]:

```
1.0
```

In [69]:

```
#training accuracy
decision_tree.score(X_test, y_test)
```

Out[69]:

```
0.7666666666666667
```

We can see overfitting, let's do post-pruning on this and check the accuracies

In [165]:

```
decision_tree = DecisionTreeClassifier(criterion = 'entropy', max_depth = 3)
```

In [166]:

```
decision_tree.fit(X_train, y_train)
```

Out[166]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=3, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

In [167]:

```
pred_decision_tree = decision_tree.predict(X_test)
pred_decision_tree
```

Out[167]:

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

In [168]:

```
#training accuracy
decision_tree.score(X_train, y_train)
```

Out[168]:

```
0.8870292887029289
```

In [192]:

```
#testing accuracy
decision_tree_accuracy = decision_tree.score(X_test, y_test)
decision_tree_accuracy
```

Out[192]:

```
0.7833333333333333
```

In [193]:

```
decision_tree_recall = recall_score(y_test, pred_decision_tree)
decision_tree_recall
```

Out[193]:

```
0.6470588235294118
```

In [194]:

```
decision_tree_precision = precision_score(y_test, pred_decision_tree)
decision_tree_precision
```

Out[194]:

```
0.6111111111111112
```

In [195]:

```
decision_tree_f1 = f1_score(y_test, pred_decision_tree)
decision_tree_f1
```

Out[195]:

```
0.6285714285714287
```

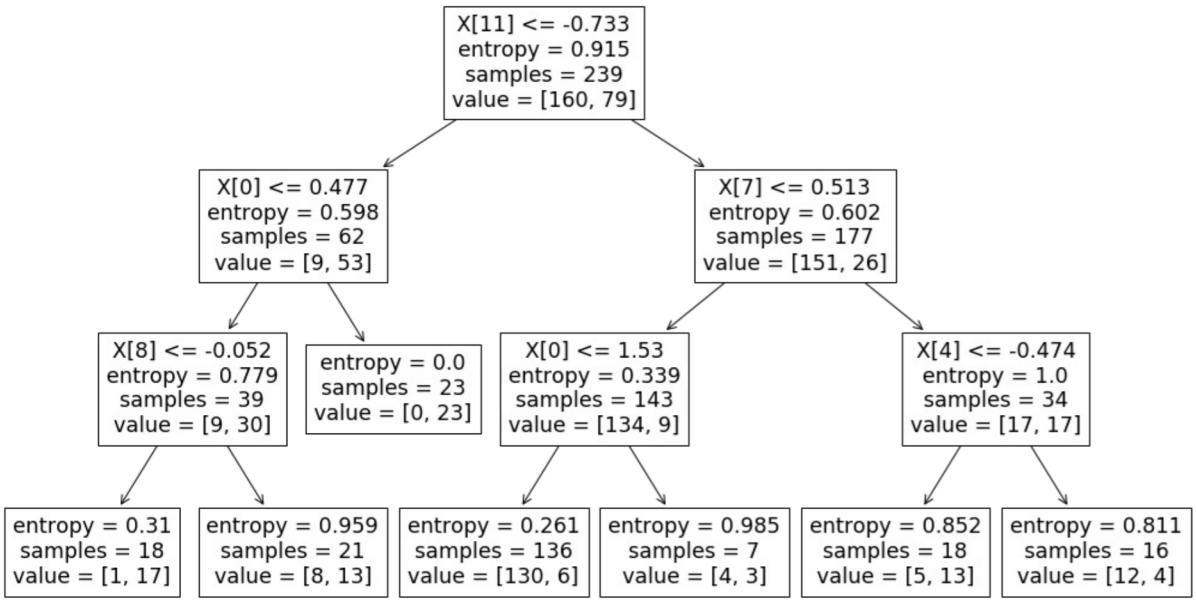
In [77]:

```
from sklearn.tree import plot_tree
```

In [78]:

```
plt.figure(figsize = (18,10))
print(plot_tree(decision_tree))
```

```
[Text(460.35, 475.65000000000003, 'X[11] <= -0.733\nentropy = 0.915\nsamples = 239\nvalue = [160, 79
]'), Text(251.10000000000002, 339.75, 'X[0] <= 0.477\nentropy = 0.598\nsamples = 62\nvalue = [9, 53]
'), Text(167.4, 203.85000000000002, 'X[8] <= -0.052\nentropy = 0.779\nsamples = 39\nvalue = [9, 30]'
), Text(83.7, 67.94999999999999, 'entropy = 0.31\nsamples = 18\nvalue = [1, 17]'), Text(251.10000000
000002, 67.94999999999999, 'entropy = 0.959\nsamples = 21\nvalue = [8, 13]'), Text(334.8, 203.850000
00000002, 'entropy = 0.0\nsamples = 23\nvalue = [0, 23]'), Text(669.6, 339.75, 'X[7] <= 0.513\nentro
py = 0.602\nsamples = 177\nvalue = [151, 26]'), Text(502.20000000000005, 203.85000000000002, 'X[0] <
= 1.53\nentropy = 0.339\nsamples = 143\nvalue = [134, 9]'), Text(418.5, 67.94999999999999, 'entropy
= 0.261\nsamples = 136\nvalue = [130, 6]'), Text(585.9, 67.94999999999999, 'entropy = 0.985\nsamples
= 7\nvalue = [4, 3]'), Text(837.0, 203.85000000000002, 'X[4] <= -0.474\nentropy = 1.0\nsamples = 34\
nvalue = [17, 17]'), Text(753.3000000000001, 67.94999999999999, 'entropy = 0.852\nsamples = 18\nvalu
e = [5, 13]'), Text(920.7, 67.94999999999999, 'entropy = 0.811\nsamples = 16\nvalue = [12, 4]')]
```



Random Forest

In [79]:

```
random_forest = RandomForestClassifier(criterion = 'entropy', max_depth = 4)
```

In [80]:

```
random forest.fit(X train, y train)
```

Out[80]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='entropy', max_depth=4, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [81]:

```
pred_random_forest = random_forest.predict(X_test)
pred random forest
```

Out[81]:

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

In [82]:

```
#training accuracy
random_forest.score(X_train, y_train)
```

Out[82]:

```
0.9163179916317992
```

In [196]:

```
#testing accuracy
random_forest_accuracy = random_forest.score(X_test, y_test)
random_forest_accuracy
```

Out[196]:

```
0.8
```

In [197]:

```
random_forest_precision = precision_score(y_test, pred_random_forest)
random_forest_precision
```

Out[197]:

```
0.6666666666666666
```

In [198]:

```
random_forest_recall = recall_score(y_test, pred_random_forest)
random_forest_recall
```

Out[198]:

```
0.5882352941176471
```

Hyperparameter Tuning

In [140]:

```
clf = RandomForestClassifier()
```

In [141]:

```
grid_param = {"n_estimators" : [10, 50, 100],
    "criterion" : ['gini','entropy'],
    'max_depth' : range(2,10),
    "max_features" : ['auto','log2'],
    "min_samples_split" : range(1,10),
    "min_impurity_decrease" : [0.0, 0.5, 1.0]
}
```

In [142]:

```
grid_search_random_forest = GridSearchCV(estimator = clf,
                          param_grid = grid_param,
                           cv = 5,
                           n_jobs = -1,
                           verbose = 3
                          )
```

In [143]:

```
grid_search_random_forest.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 2592 candidates, totalling 12960 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:    3.0s
[Parallel(n_jobs=-1)]: Done 112 tasks      | elapsed:    4.8s
[Parallel(n_jobs=-1)]: Done 600 tasks      | elapsed:   12.4s
[Parallel(n_jobs=-1)]: Done 1496 tasks      | elapsed:   27.8s
[Parallel(n_jobs=-1)]: Done 2648 tasks      | elapsed:   48.9s
[Parallel(n_jobs=-1)]: Done 4056 tasks      | elapsed:  1.2min
[Parallel(n_jobs=-1)]: Done 5289 tasks      | elapsed:  1.7min
[Parallel(n_jobs=-1)]: Done 6456 tasks      | elapsed:  2.1min
[Parallel(n_jobs=-1)]: Done 7000 tasks      | elapsed:  2.4min
[Parallel(n_jobs=-1)]: Done 8024 tasks      | elapsed:  2.8min
[Parallel(n_jobs=-1)]: Done 9368 tasks      | elapsed:  3.3min
[Parallel(n_jobs=-1)]: Done 10840 tasks      | elapsed:  3.9min
[Parallel(n_jobs=-1)]: Done 12440 tasks      | elapsed:  4.6min
[Parallel(n_jobs=-1)]: Done 12945 out of 12960 | elapsed:  4.8min remaining:    0.2s
[Parallel(n_jobs=-1)]: Done 12960 out of 12960 | elapsed:  4.8min finished
```

Out[143]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=100, n_jobs=None,...
                                              random_state=None, verbose=0,
                                              warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(2, 10),
                         'max_features': ['auto', 'log2'],
                         'min_impurity_decrease': [0.0, 0.5, 1.0],
                         'min_samples_split': range(1, 10),
                         'n_estimators': [10, 50, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=3)
```

In [144]:

```
grid_search_random_forest.best_params_
```

Out[144]:

```
{'criterion': 'gini',
 'max_depth': 6,
 'max_features': 'log2',
 'min_impurity_decrease': 0.0,
 'min_samples_split': 9,
 'n_estimators': 50}
```

In [146]:

```
random_forest = RandomForestClassifier( n_estimators = 50,
            criterion = 'gini',
            max_depth = 6,
            max_features = "log2",
            min_impurity_decrease = 0.0,
            min_samples_split = 9
)
```

In [147]:

```
random_forest.fit(X_train, y_train)
```

Out[147]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=6, max_features='log2',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=9,
                       min_weight_fraction_leaf=0.0, n_estimators=50,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [148]:

```
randon_forest_pred = random_forest.predict(X_test)
randon_forest_pred
```

Out[148]:

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

In [149]:

```
#training accuracy
random_forest.score(X_train, y_train)
```

Out[149]:

```
0.9456066945606695
```

In [199]:

```
#testing accuracy
random_forest_accuracy = random_forest.score(X_test, y_test)
random_forest_accuracy
```

Out[199]:

```
0.8
```

In [200]:

```
random_forest_precision = precision_score(y_test, randon_forest_pred)
random_forest_precision
```

Out[200]:

```
0.6470588235294118
```

In [201]:

```
random_forest_recall = recall_score(y_test, randon_forest_pred)
random_forest_recall
```

Out[201]:

```
0.6470588235294118
```

In [203]:

```
random_forest_f1 = f1_score(y_test, randon_forest_pred)
random_forest_f1
```

Out[203]:

```
0.6470588235294118
```

Results

In [204]:

```
logistic_accuracy
```

Out[204]:

```
0.7666666666666667
```

In [205]:

```
knn_accuracy
```

Out[205]:

```
0.75
```

In [206]:

```
kmeans_accuracy
```

Out[206]:

```
0.5
```

In [207]:

```
xgboost_accuracy
```

Out[207]:

```
0.8333333333333334
```

In [208]:

```
decision_tree_accuracy
```

Out[208]:

```
0.7833333333333333
```

In [209]:

```
random_forest_accuracy
```

Out[209]:

```
0.8
```

In [210]:

```
Results = pd.DataFrame()
```

In [211]:

```
Results["Accuracy"] = [logistic_accuracy, knn_accuracy, kmeans_accuracy, xgboost_accuracy, decision_tree_accuracy
, random_forest_accuracy]
Results["Precision"] = [logistic_precision, knn_precision, kmeans_precision, xgboost_precision, decision_tree_pre
cision, random_forest_precision]
Results["Recall"] = [logistic_recall, knn_recall, kmeans_recall, xgboost_recall, decision_tree_recall, random_for
est_recall]
Results["F1 Score"] = [logistic_f1, knn_f1, kmeans_f1, xgboost_f1, decision_tree_f1, random_forest_f1]
```

In [213]:

```
models = ["Logistic Regression", "K-Nearest Neighbor", "K Means Clustering", "XGBoost", "Decision Tree", "Random
Forest"]
models
```

Out[213]:

```
['Logistic Regression',
 'K-Nearest Neighbor',
 'K Means Clustering',
 'XGBoost',
 'Decision Tree',
 'Random Forest']
```

In [221]:

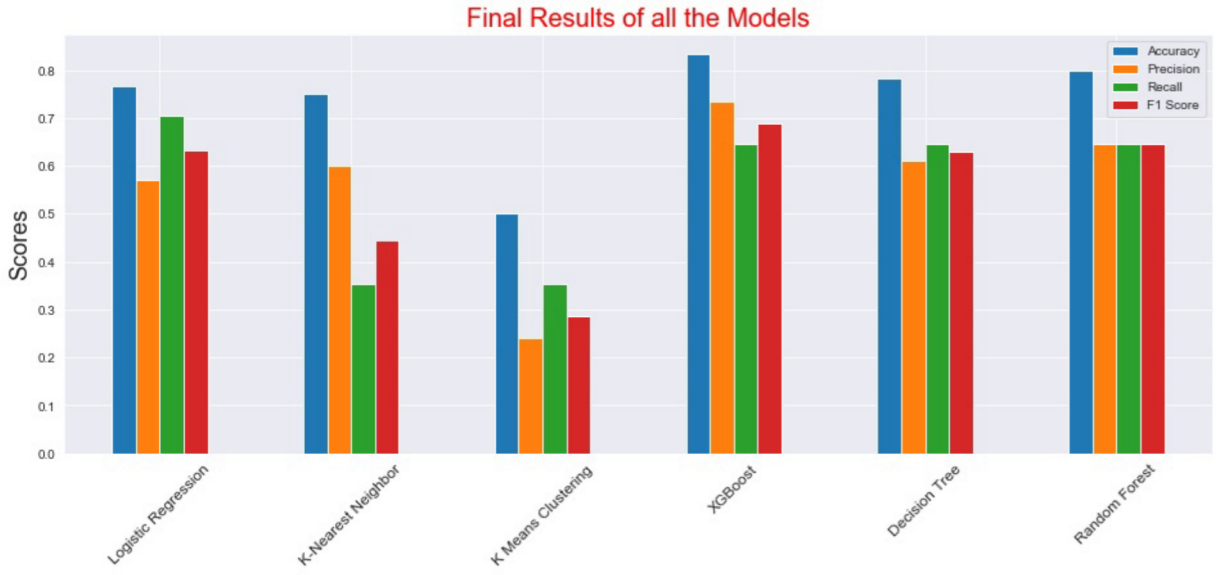
```
Results.index = models
round(Results, 3)
```

Out[221]:

	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.767	0.571	0.706	0.632
K-Nearest Neighbor	0.750	0.600	0.353	0.444
K Means Clustering	0.500	0.240	0.353	0.286
XGBoost	0.833	0.733	0.647	0.688
Decision Tree	0.783	0.611	0.647	0.629
Random Forest	0.800	0.647	0.647	0.647

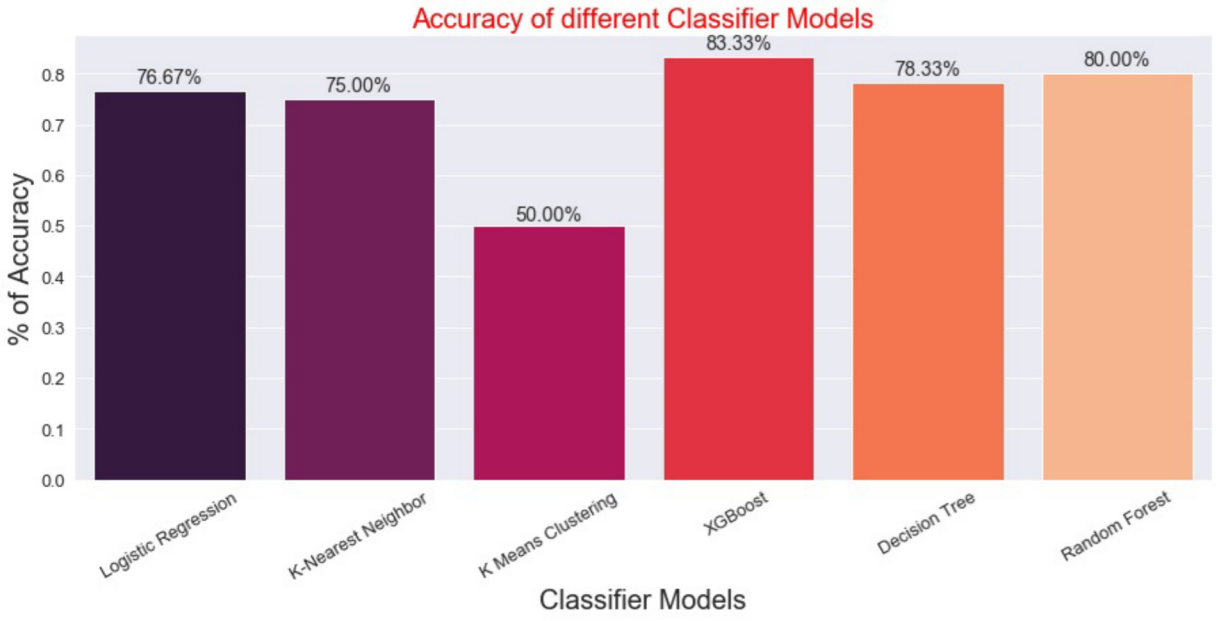
In [353]:

```
plt.rcParams['figure.figsize'] = 16,6
Results.plot(kind = "bar")
plt.legend(Results.columns)
plt.title("Final Results of all the Models", fontsize = 20, color = "red")
plt.xticks(rotation = 45, fontsize = 12)
plt.ylabel("Scores", fontsize = 18)
plt.show()
```



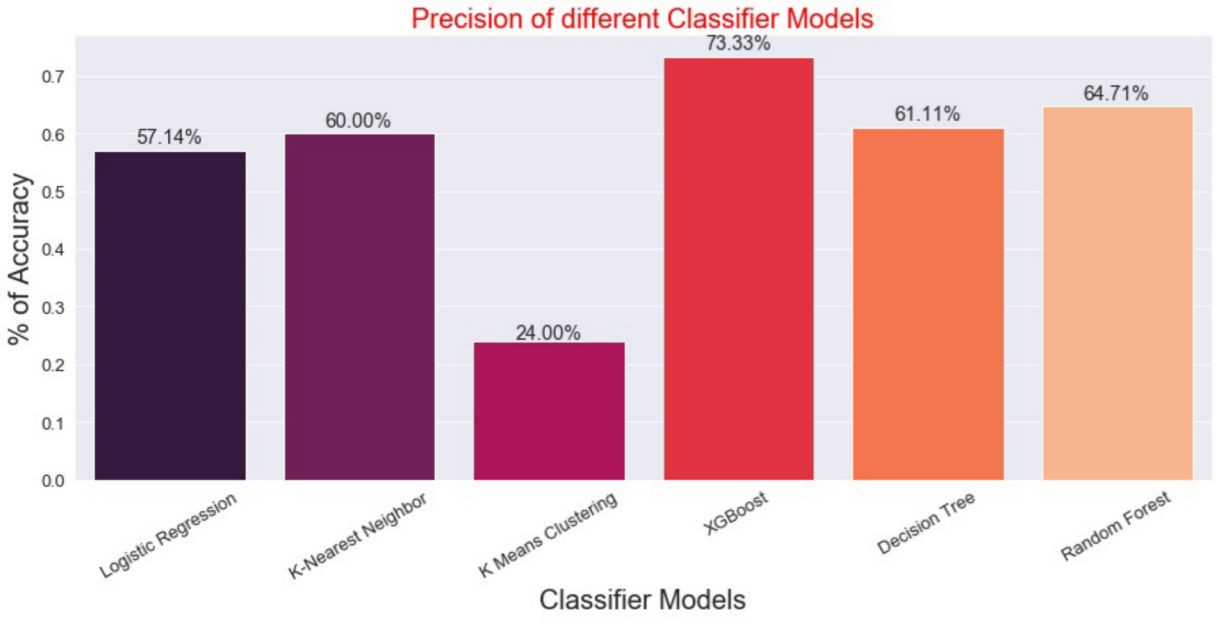
In [356]:

```
plt.rcParams['figure.figsize'] = 15,6
sns.set_style("darkgrid")

ax = sns.barplot(x = models, y = Results["Accuracy"], palette = "rocket", saturation = 1.5)
plt.xlabel("Classifier Models", fontsize = 20 )
plt.ylabel("% of Accuracy", fontsize = 20)
plt.title("Accuracy of different Classifier Models", fontsize = 20, color = "red")
plt.xticks(fontsize = 13, horizontalalignment = 'center', rotation = 30)
plt.yticks(fontsize = 13)

for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    ax.annotate(f'{height:.2%}', (x + width/2, y + height*1.02), ha='center', fontsize = 'x-large')
plt.show()
```



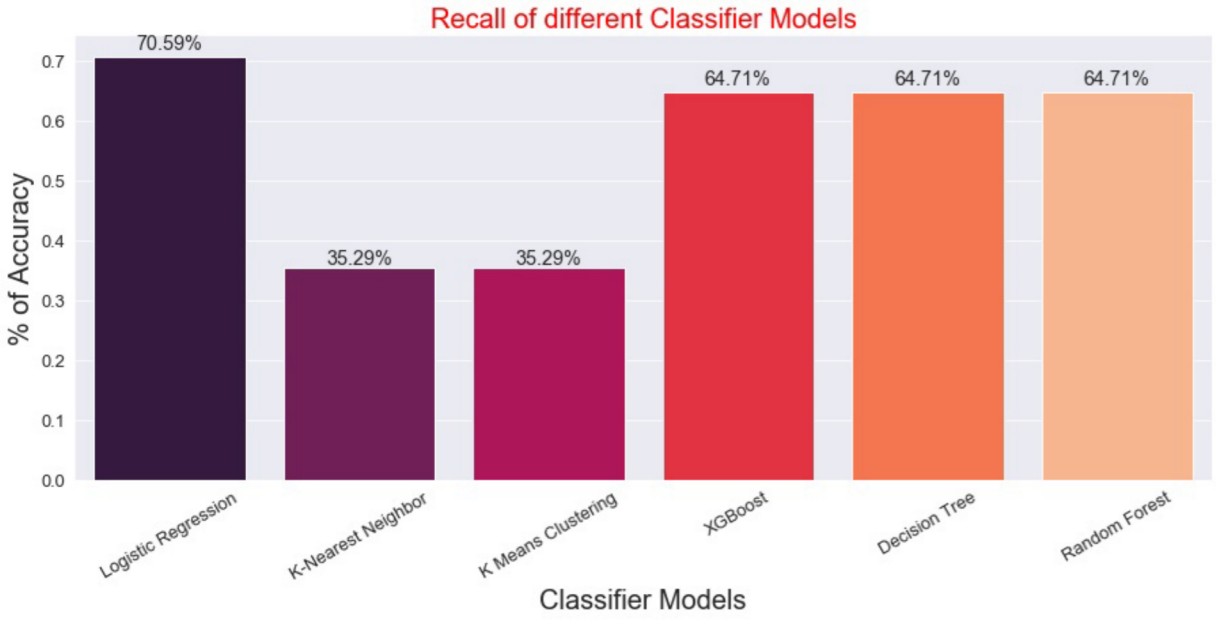
In [345]:

```
plt.rcParams['figure.figsize'] = 15,6
sns.set_style("darkgrid")

ax = sns.barplot(x = models, y = Results["Precision"], palette = "rocket", saturation = 1.5)
plt.xlabel("Classifier Models", fontsize = 20 )
plt.ylabel("% of Accuracy", fontsize = 20)
plt.title("Precision of different Classifier Models", fontsize = 20, color = "red")
plt.xticks(fontsize = 13, horizontalalignment = 'center', rotation = 30)
plt.yticks(fontsize = 13)

for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    ax.annotate(f'{height:.2%}', (x + width/2, y + height*1.02), ha='center', fontsize = 'x-large')
plt.show()
```



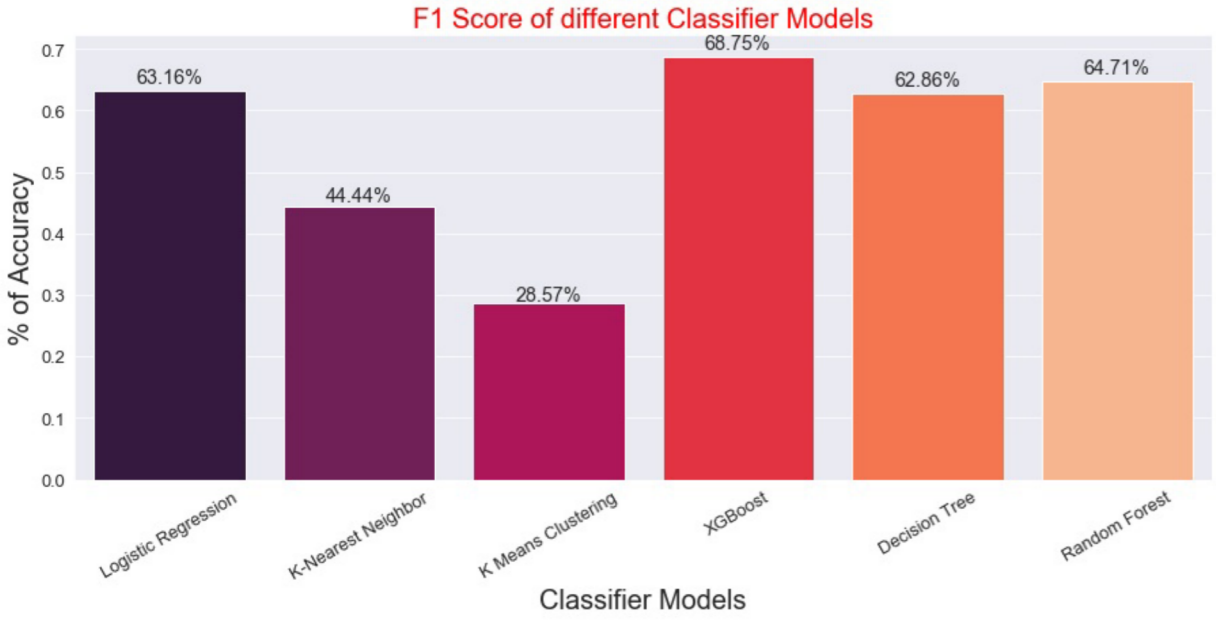
In [250]:

```
plt.rcParams['figure.figsize'] = 15,6
sns.set_style("darkgrid")

ax = sns.barplot(x = models, y = Results["Recall"], palette = "rocket", saturation = 1.5)
plt.xlabel("Classifier Models", fontsize = 20 )
plt.ylabel("% of Accuracy", fontsize = 20)
plt.title("Recall of different Classifier Models", fontsize = 20, color = "red")
plt.xticks(fontsize = 13, horizontalalignment = 'center', rotation = 30)
plt.yticks(fontsize = 13)

for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    ax.annotate(f'{height:.2%}', (x + width/2, y + height*1.02), ha='center', fontsize = 'x-large')
plt.show()
```



In [251]:

```
plt.rcParams['figure.figsize'] = 15,6
sns.set_style("darkgrid")

ax = sns.barplot(x = models, y = Results["F1 Score"], palette = "rocket", saturation = 1.5)
plt.xlabel("Classifier Models", fontsize = 20 )
plt.ylabel("% of Accuracy", fontsize = 20)
plt.title("F1 Score of different Classifier Models", fontsize = 20, color = "red")
plt.xticks(fontsize = 13, horizontalalignment = 'center', rotation = 30)
plt.yticks(fontsize = 13)

for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    ax.annotate(f'{height:.2%}', (x + width/2, y + height*1.02), ha='center', fontsize = 'x-large')
plt.show()
```



In [ ]:

INTERPRETATION

This is a classic case of a Binary Classification. We need to predict whether a patient would die of Heart Attack or not. I used six different classification algorithms, which are namely Logistic Regression, KNN, K Means Clustering, XGBoost, Decision Tree Random Forest, and compared their results using different parameters.

Firstly, we can surely compare the accuracies of all the models. Accuracy is given as:

(TP + TN) / (TP + TN + FP + FN)

This measure tells us that how many true and how many false were correctly predicted among all the predictions. Accuracy wise, **XGBoost** performed the best and **Random Forest** wasn't far behind as well.

But it is extremely important in this dataset that we measure the Recall or the True Positive Rate, whose formula is:

TP/ (TP + FN)

This tells us that out of all the patients who had a heart attack and died because of it, how many were actually detected, which is even more important that finding out the overall accuracy in this problem.

If we compare the Recall/Sensitivity/True Positive Rate of all the models, we can notice that the **Logistic Regression** performed the best among all the six models.

So, if I were to choose a model for this problem, I would go with the **Logistic Regression** because in my opinion Recall is more important in this classification problem that the overall accuracy of the models.