# Healthcare Project-02

March 26, 2022

# 1 Name: Sunil Pradhan

## 1.1 DOMAIN-HEALTHCARE

## 1.2 OBJECTIVE-

1.NIDDK (National Institute of Diabetes and Digestive and Kidney Diseases) research creates knowledge about and treatments for the most chronic, costly, and consequential diseases.

2.The dataset used in this project is originally from NIDDK. The objective is to predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset.

3.Build a model to accurately predict whether the patients in the dataset have diabetes or not.

## 1.3 Project Task: Week 1(Data Exploration)

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

[2]:
```
df=pd.read_csv("health care diabetes.csv")
df.head()
```

[2]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
```

```
4                    2.288   33        1
```

[3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

[4]:
```
df.describe()
```

[4]:
```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

[5]:
```
df.columns
```

[5]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
```

```
dtype='object')
```

[6]:
```
df[df["Glucose"]==0]
```

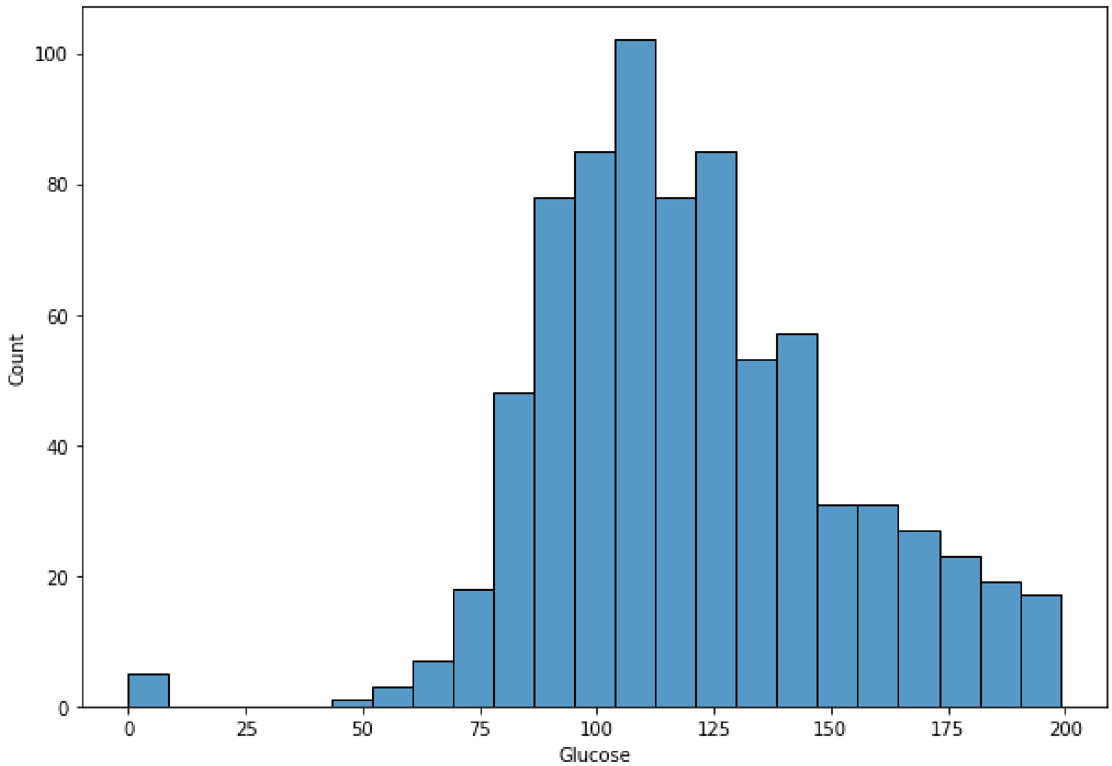
[6]:
```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
75             1        0             48             20        0  24.7
182            1        0             74             20       23  27.7
342            1        0             68             35        0  32.0
349            5        0             80             32        0  41.0
502            6        0             68             41        0  39.0

     DiabetesPedigreeFunction  Age  Outcome
75                      0.140   22        0
182                     0.299   21        0
342                     0.389   22        0
349                     0.346   37        1
502                     0.727   41        1
```

[7]:
```
df["Glucose"].value_counts()
```

[7]:
```
99     17
100    17
111    14
129    14
125    14
       ..
191     1
177     1
44      1
62      1
190     1
Name: Glucose, Length: 136, dtype: int64
```

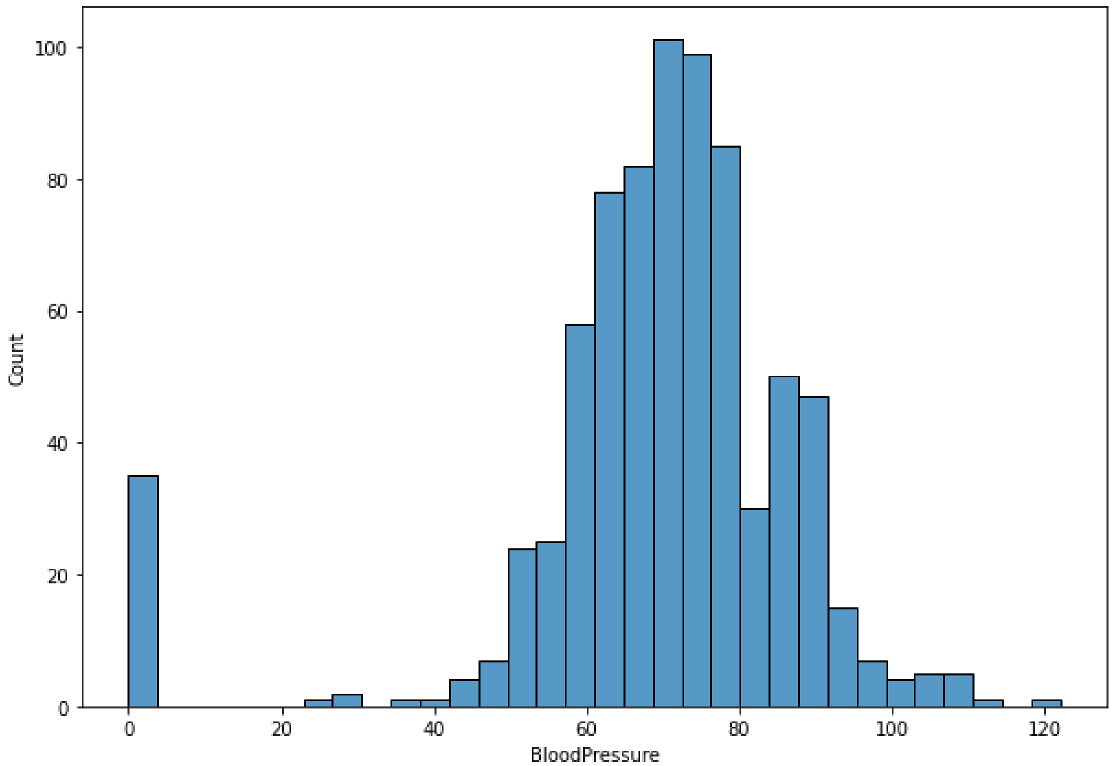
[8]:
```
plt.figure(figsize=(10,7))
sns.histplot(df["Glucose"])
```

[8]:
```
<AxesSubplot:xlabel='Glucose', ylabel='Count'>
```



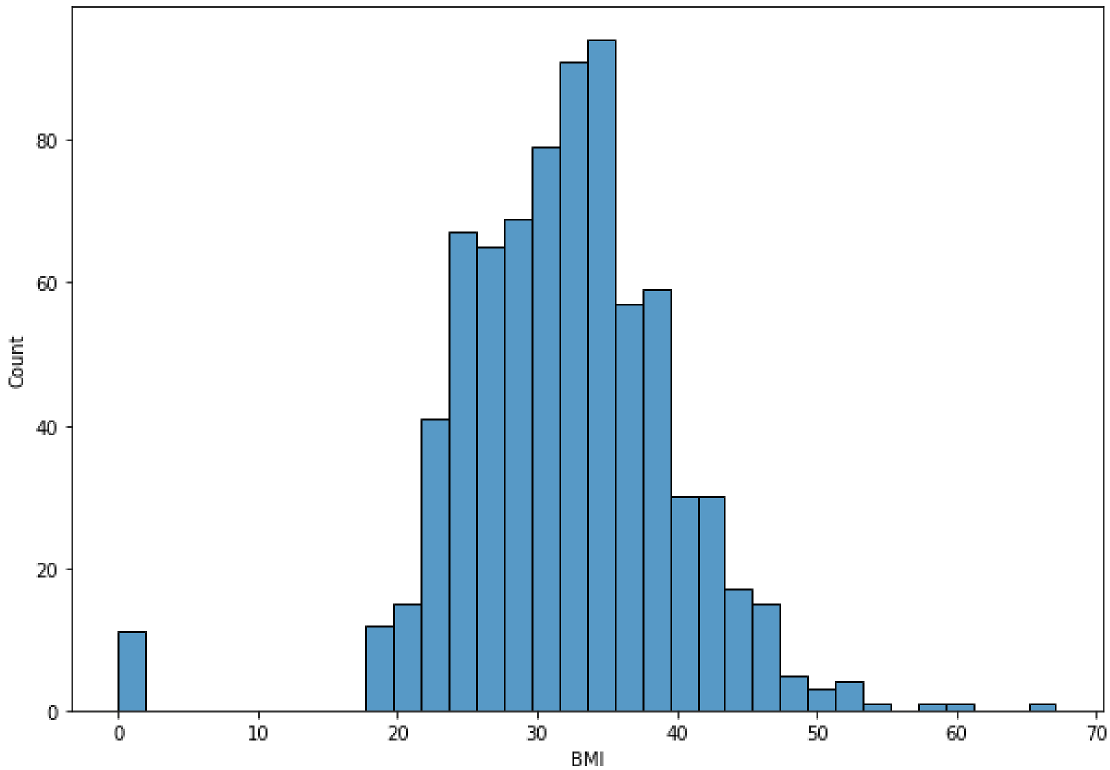

[9]:
```
plt.figure(figsize=(10,7))
sns.histplot(df["BloodPressure"])
```

[9]: <AxesSubplot:xlabel='BloodPressure', ylabel='Count'>



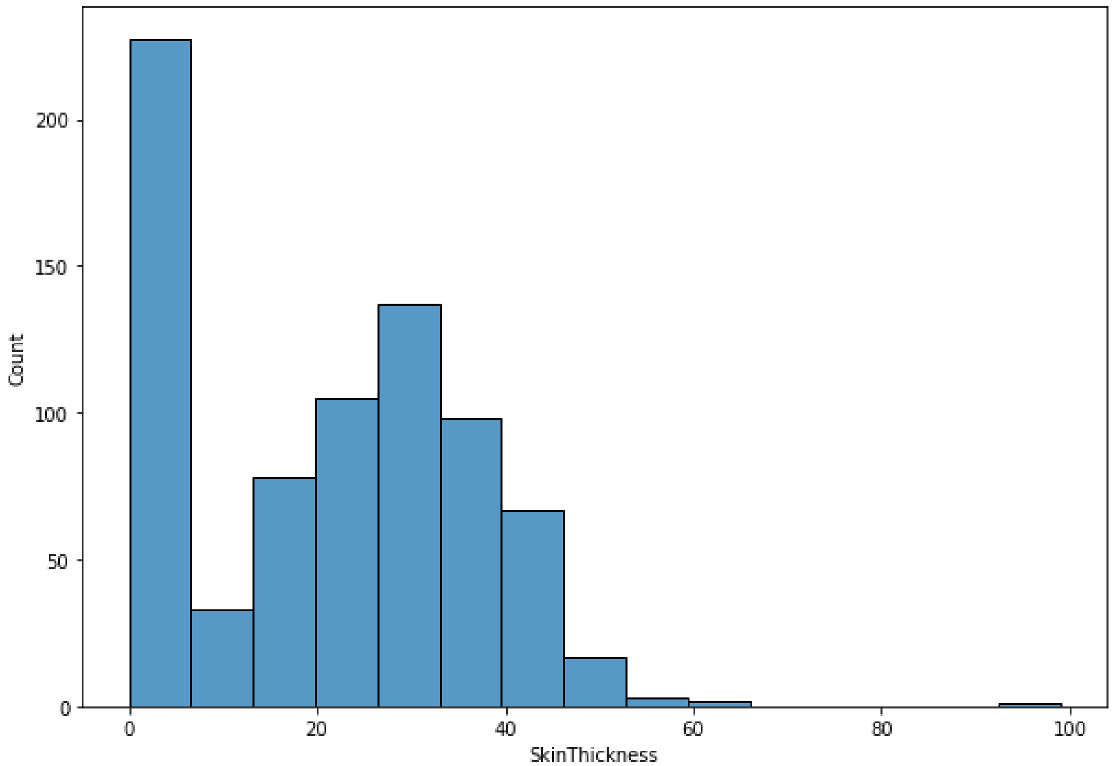

[10]:
```
plt.figure(figsize=(10,7))
sns.histplot(df["BMI"])
```

[10]: <AxesSubplot:xlabel='BMI', ylabel='Count'>



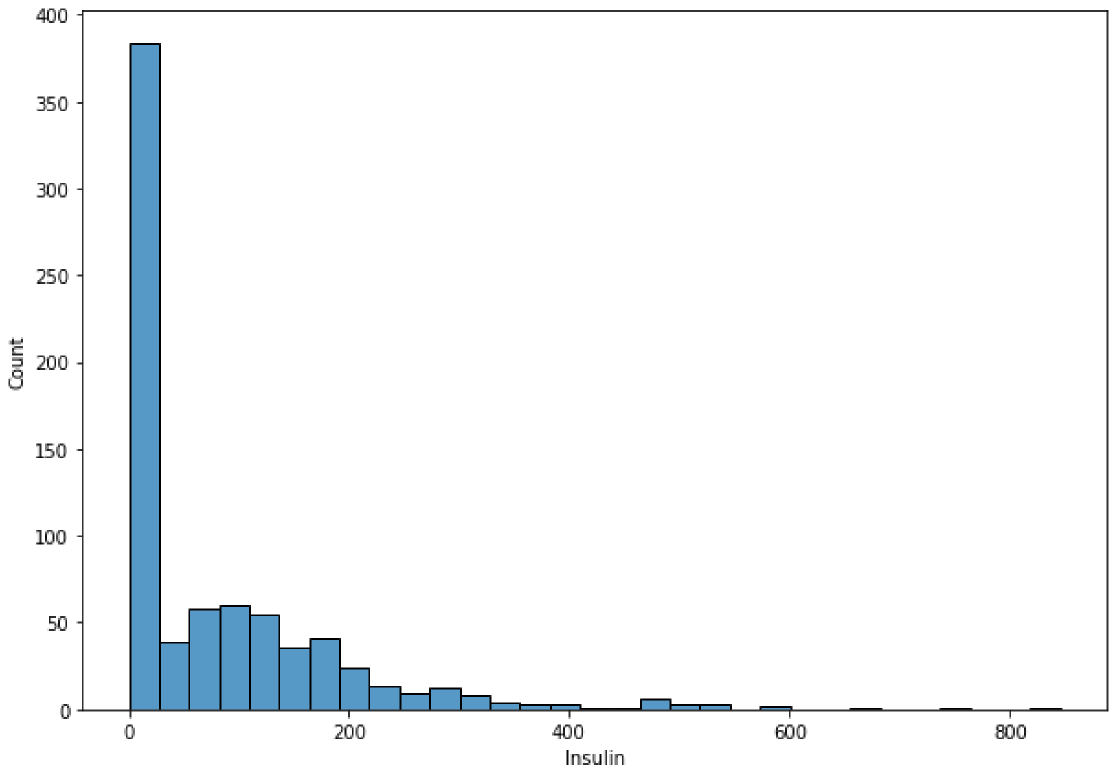

```
[11]: plt.figure(figsize=(10,7))
      sns.histplot(df["SkinThickness"])
```

```
[11]: <AxesSubplot:xlabel='SkinThickness', ylabel='Count'>
```




[12]:
```
plt.figure(figsize=(10,7))
sns.histplot(df["Insulin"])
```

[12]: <AxesSubplot:xlabel='Insulin', ylabel='Count'>




[13]:
```
df["Glucose"].mean()
```

[13]: 120.89453125

### 1.3.1 Repalce missing values i.e. 0

[14]:
```
df["Glucose"]=df["Glucose"].replace(0,df["Glucose"].median())
df["BMI"]=df["BMI"].replace(0,df["BMI"].mean())
df["BloodPressure"]=df["BloodPressure"].replace(0,df["BloodPressure"].median())
df["Insulin"]=df["Insulin"].replace(0,df["Insulin"].median())
df["SkinThickness"]=df["SkinThickness"].replace(0,df["SkinThickness"].median())
```

[15]:
```
df.head()
```

[15]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35     30.5  33.6
1            1       85             66             29     30.5  26.6
2            8      183             64             23     30.5  23.3
3            1       89             66             23     94.0  28.1
4            0      137             40             35    168.0  43.1

   DiabetesPedigreeFunction  Age  Outcome
```

```
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

[16]:
```
df.tail()
```

[16]:
```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
763           10      101             76             48    180.0  32.9
764            2      122             70             27     30.5  36.8
765            5      121             72             23    112.0  26.2
766            1      126             60             23     30.5  30.1
767            1       93             70             31     30.5  30.4

     DiabetesPedigreeFunction  Age  Outcome
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0
```

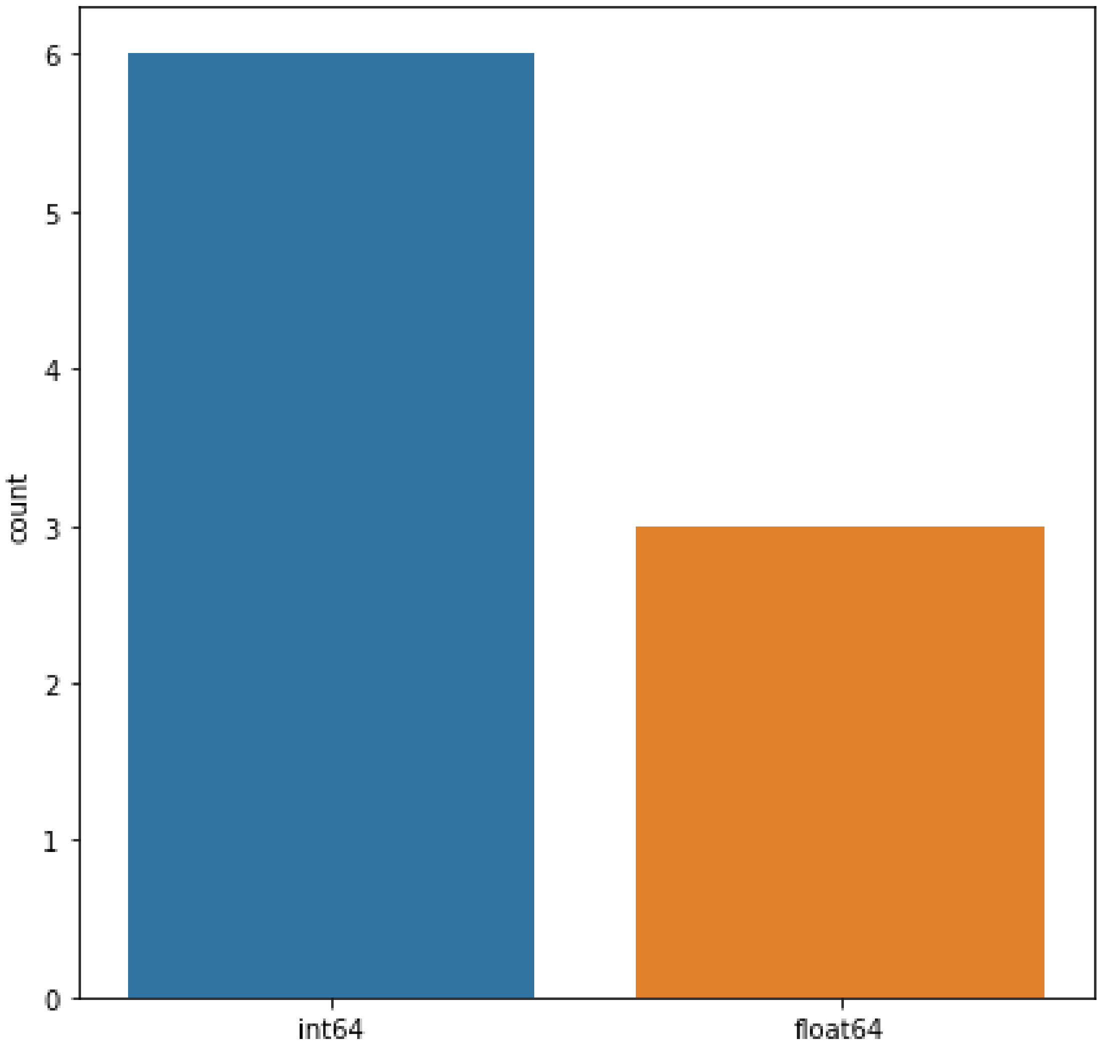
#### 1.3.2 There are integer and float data type variables in this dataset. Create a count (frequency) plot describing the data types and the count of variables

[17]:
```
df.dtypes.value_counts()
```

[17]:
```
int64      6
float64    3
dtype: int64
```

[18]:
```
plt.figure(figsize=(7,7))
sns.countplot(df.dtypes)
```

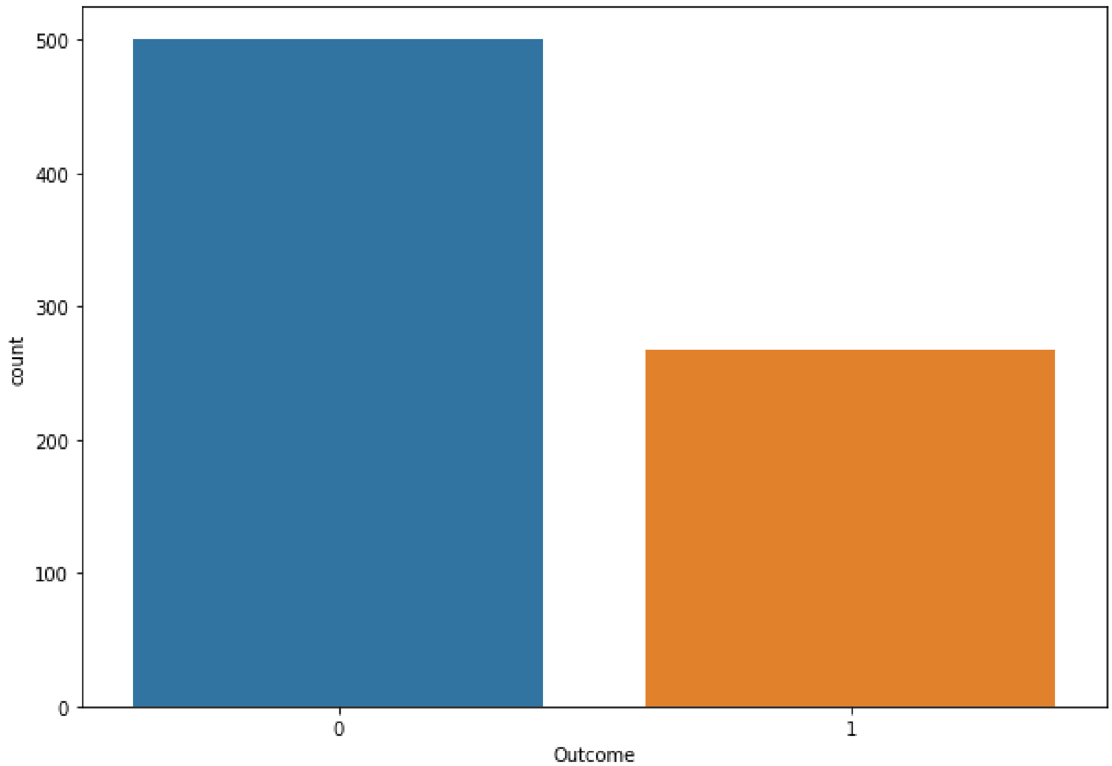
[18]:
```
<AxesSubplot:ylabel='count'>
```




#### 1.3.3 Check the balance of the data by plotting the count of outcomes by their value. Describe your findings and plan future course of action

[19]:
```
plt.figure(figsize=(10,7))
sns.countplot(df["Outcome"])
```

[19]: <AxesSubplot:xlabel='Outcome', ylabel='count'>

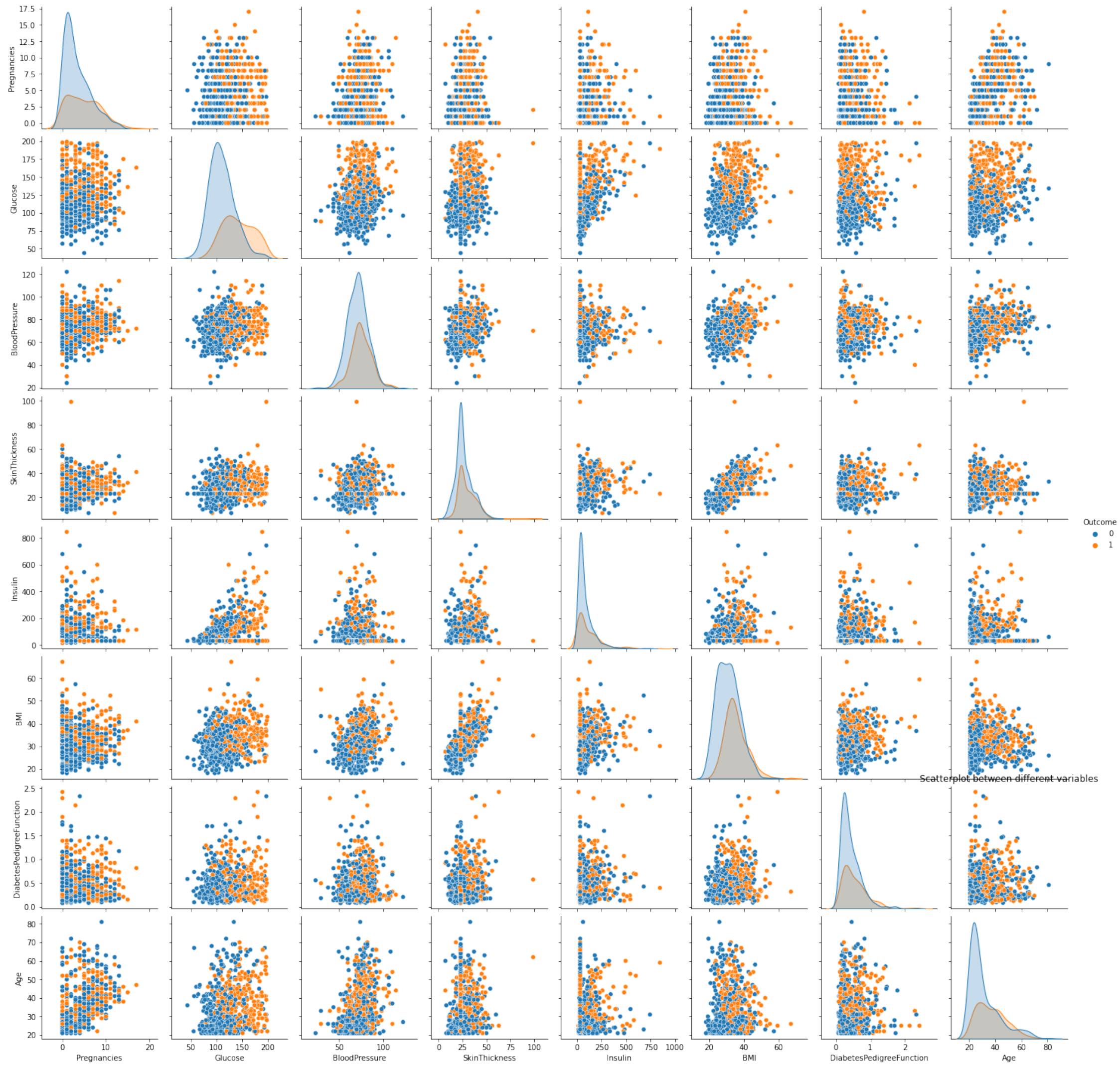



**Since, the dataset is balanced.Thus, we don't apply random sampling tachnique for balancing the dataset.**

### 1.4 Create scatter charts between the pair of variables to understand the relationships

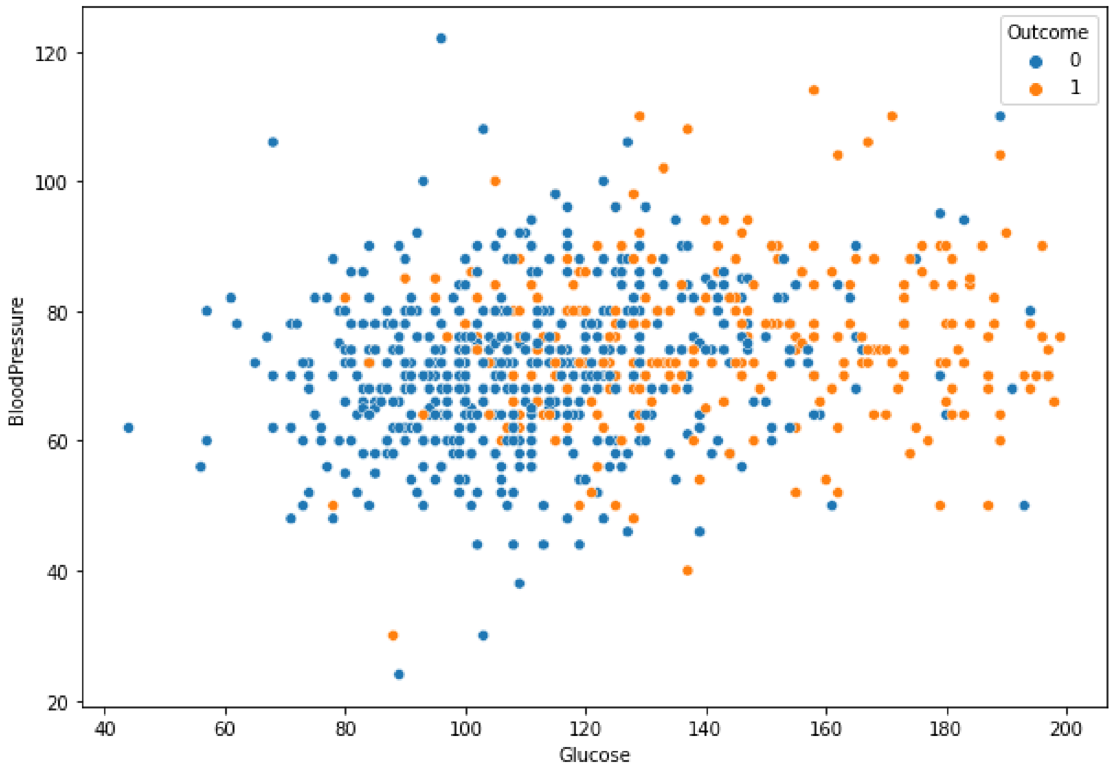
[20]:
```
sns.pairplot(data=df,hue="Outcome")
plt.title("Scatterplot between different variables")
```

[20]: Text(0.5, 1.0, 'Scatterplot between different variables')




```
[21]: plt.figure(figsize=(10,7))
      sns.scatterplot("Glucose","BloodPressure",data=df,hue="Outcome")
```

[21]: <AxesSubplot:xlabel='Glucose', ylabel='BloodPressure'>




### 1.5 Perform correlation analysis. Visually explore it using a heat map

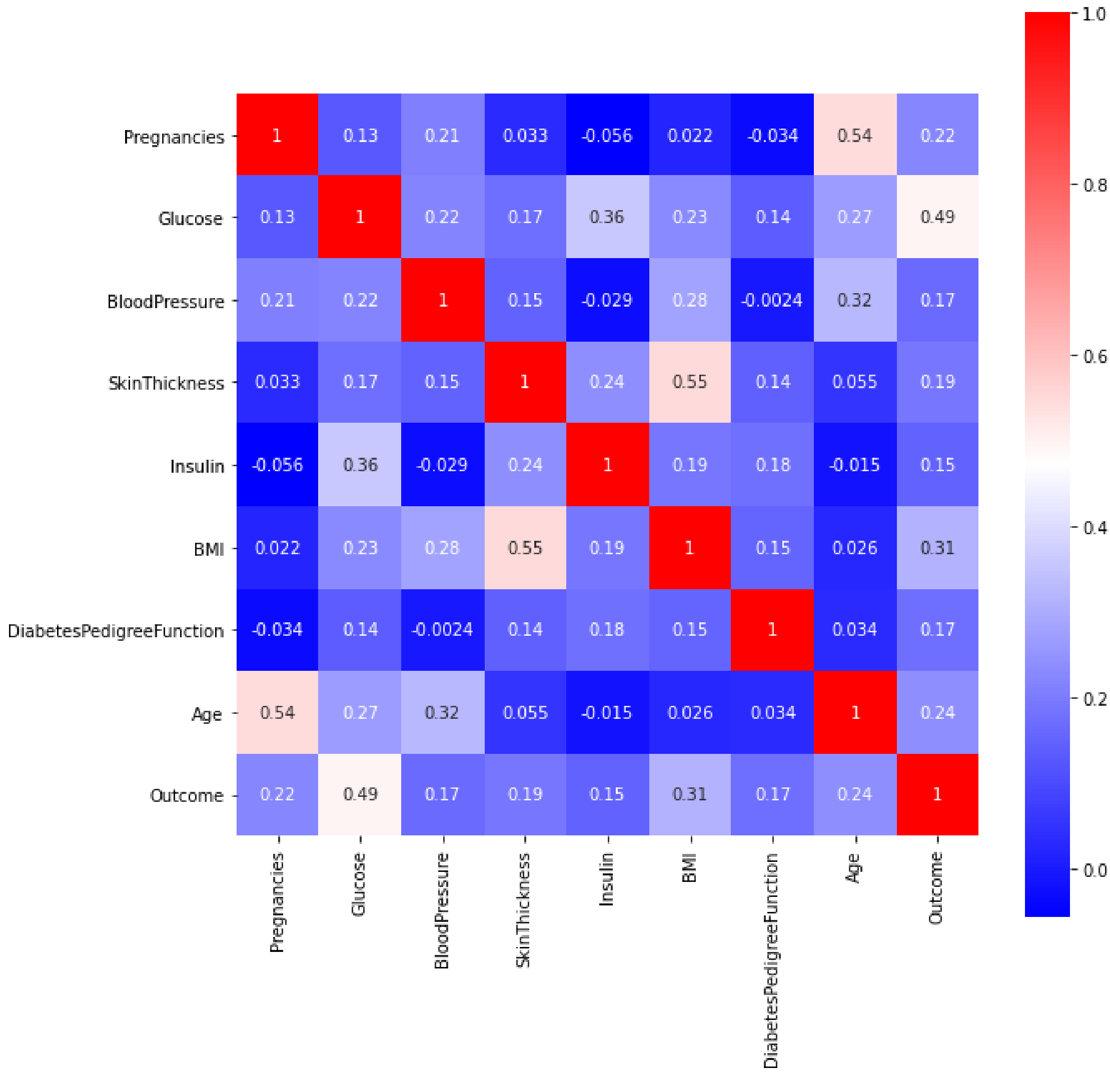
[22]:
```
df.corr()
```

[22]:
```
                          Pregnancies   Glucose  BloodPressure  SkinThickness  \
Pregnancies                  1.000000  0.128213       0.208615       0.032568
Glucose                      0.128213  1.000000       0.218937       0.172143
BloodPressure                0.208615  0.218937       1.000000       0.147809
SkinThickness                0.032568  0.172143       0.147809       1.000000
Insulin                     -0.055697  0.357573      -0.028721       0.238188
BMI                          0.021546  0.231408       0.281129       0.546958
DiabetesPedigreeFunction    -0.033523  0.137327      -0.002378       0.142977
Age                          0.544341  0.266909       0.324915       0.054514
Outcome                      0.221898  0.492782       0.165723       0.189065

                           Insulin       BMI  DiabetesPedigreeFunction  \
Pregnancies              -0.055697  0.021546                 -0.033523
Glucose                   0.357573  0.231408                  0.137327
BloodPressure            -0.028721  0.281129                 -0.002378
SkinThickness             0.238188  0.546958                  0.142977
Insulin                   1.000000  0.189031                  0.178029
BMI                       0.189031  1.000000                  0.153508
```

```
DiabetesPedigreeFunction  0.178029  0.153508                  1.000000
Age                      -0.015413  0.025748                  0.033561
Outcome                   0.148457  0.312254                  0.173844

                               Age   Outcome
Pregnancies               0.544341  0.221898
Glucose                   0.266909  0.492782
BloodPressure             0.324915  0.165723
SkinThickness             0.054514  0.189065
Insulin                  -0.015413  0.148457
BMI                       0.025748  0.312254
DiabetesPedigreeFunction  0.033561  0.173844
Age                       1.000000  0.238356
Outcome                   0.238356  1.000000
```

[23]:
```
plt.figure(figsize=(10,10))
sns.heatmap(df.corr(),cmap="bwr",square=True,annot=True)
```

[23]:
```
<AxesSubplot:>
```




### 1.6 Project Task: Week 2(Data modeling)

1.Devise strategies for model building. It is important to decide the right validation framework. Express your thought process.

2.Apply an appropriate classification algorithm to build a model.

3.Compare various models with the results from KNN algorithm.

4.Create a classification report by analyzing sensitivity, specificity, AUC (ROC curve), etc.

[24]:
```
df.head()
```

[24]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35     30.5  33.6
1            1       85             66             29     30.5  26.6
```

```
2            8      183             64              23     30.5  23.3
3            1       89             66              23     94.0  28.1
4            0      137             40              35    168.0  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

[25]:
```
x=df.drop("Outcome",axis=1)
y=df[["Outcome"]]
```

[26]:
```
x.head()
```

[26]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72              35     30.5  33.6
1            1       85             66              29     30.5  26.6
2            8      183             64              23     30.5  23.3
3            1       89             66              23     94.0  28.1
4            0      137             40              35    168.0  43.1

   DiabetesPedigreeFunction  Age
0                     0.627   50
1                     0.351   31
2                     0.672   32
3                     0.167   21
4                     2.288   33
```

[27]:
```
y.head()
```

[27]:
```
   Outcome
0        1
1        0
2        1
3        0
4        1
```

[28]:
```
from sklearn.model_selection import train_test_split
```

[29]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.
 ↪25,random_state=25)
```

[30]:
```
x_train.shape,x_test.shape
```

[30]:
```
((576, 8), (192, 8))
```

[31]:
```
y_train.shape,y_test.shape
```

[31]:
```
((576, 1), (192, 1))
```

[32]:
```
x_train.head()
```

[32]:
```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
427            1      181             64             30    180.0  34.1
133            8       84             74             31     30.5  38.3
262            4       95             70             32     30.5  32.1
358           12       88             74             40     54.0  35.3
489            8      194             80             23     30.5  26.1

     DiabetesPedigreeFunction  Age
427                     0.328   38
133                     0.457   39
262                     0.612   24
358                     0.378   48
489                     0.551   67
```

[33]:
```
from sklearn.preprocessing import StandardScaler
```

[34]:
```
scaler=StandardScaler()
x_train=scaler.fit_transform(x_train)
x_test=scaler.transform(x_test)
```

[35]:
```
x_train
```

[35]:
```
array([[-0.82827947,  1.9049318 , -0.72994462, …,  0.2534439 ,
        -0.42508999,  0.37784439],
       [ 1.24216246, -1.24181301,  0.10494041, …,  0.86654809,
        -0.0415753 ,  0.46339407],
       [ 0.05905278, -0.88496566, -0.2290136 , …, -0.03851047,
         0.41923693, -0.81985104],
       …,
       [ 0.05905278, -0.26859296, -0.72994462, …, -0.50563747,
        -1.02563238, -0.81985104],
       [-0.23672464, -0.23615229, -0.56296761, …,  0.83735265,
        -0.95428081, -0.47765234],
       [-0.23672464,  1.54808445, -0.72994462, …,  0.31183478,
        -0.34184649, -0.306553  ]])
```

## 1.7 Logistic regression

[36]:
```
from sklearn.linear_model import LogisticRegression
```

[37]:
```
log_model=LogisticRegression()
log_model.fit(x_train,y_train)
```

[37]:
```
LogisticRegression()
```

[38]:
```
log_pred=log_model.predict(x_test)
log_pred
```

[38]:
```
array([1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1,
       1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1], dtype=int64)
```

[39]:
```
from sklearn.metrics import␣
↪accuracy_score,confusion_matrix,classification_report,roc_curve,auc,RocCurveDisplay
```

[40]:
```
accuracy_score(log_pred,y_test)
```

[40]:
```
0.8020833333333334
```

[41]:
```
accuracy_score(log_model.predict(x_train),y_train)
```

[41]:
```
0.765625
```

[42]:
```
confusion_matrix(log_pred,y_test)
```

[42]:
```
array([[115,  23],
       [ 15,  39]], dtype=int64)
```

### 1.8 Sensitivity=TP/(TP+FN)

### 1.9 Specificity=TN/(TN+FP)

[43]:
```
print("Sensitivity is ",115/(115+39))
print("Specificity is ",23/(23+15))
```

```
Sensitivity is  0.7467532467532467
Specificity is  0.6052631578947368
```

[44]:
```
print(classification_report(log_pred,y_test))
```

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

```
              0       0.88      0.83      0.86       138
              1       0.63      0.72      0.67        54

       accuracy                           0.80       192
      macro avg       0.76      0.78      0.77       192
   weighted avg       0.81      0.80      0.81       192
```

[45]:
```
log_model.predict_proba(x_test)
```

[45]:
```
array([[0.49718064, 0.50281936],
       [0.5184736 , 0.4815264 ],
       [0.63650222, 0.36349778],
       [0.23770204, 0.76229796],
       [0.68861673, 0.31138327],
       [0.71208456, 0.28791544],
       [0.9387175 , 0.0612825 ],
       [0.79443084, 0.20556916],
       [0.92992385, 0.07007615],
       [0.90652177, 0.09347823],
       [0.33810021, 0.66189979],
       [0.85035462, 0.14964538],
       [0.70225282, 0.29774718],
       [0.48135433, 0.51864567],
       [0.77123986, 0.22876014],
       [0.30883256, 0.69116744],
       [0.81033777, 0.18966223],
       [0.87974871, 0.12025129],
       [0.71237312, 0.28762688],
       [0.64732719, 0.35267281],
       [0.80837603, 0.19162397],
       [0.89390753, 0.10609247],
       [0.92169482, 0.07830518],
       [0.57197108, 0.42802892],
       [0.9186062 , 0.0813938 ],
       [0.66550181, 0.33449819],
       [0.67938683, 0.32061317],
       [0.238574  , 0.761426  ],
       [0.78020378, 0.21979622],
       [0.83503884, 0.16496116],
       [0.66585115, 0.33414885],
       [0.35056697, 0.64943303],
       [0.87144477, 0.12855523],
       [0.12191586, 0.87808414],
       [0.4654908 , 0.5345092 ],
       [0.75957272, 0.24042728],
       [0.11965296, 0.88034704],
```

```
[0.60990487, 0.39009513],
[0.68606143, 0.31393857],
[0.89322221, 0.10677779],
[0.51946661, 0.48053339],
[0.92167769, 0.07832231],
[0.33351575, 0.66648425],
[0.66547305, 0.33452695],
[0.98254506, 0.01745494],
[0.8455861 , 0.1544139 ],
[0.77630737, 0.22369263],
[0.91105057, 0.08894943],
[0.84328445, 0.15671555],
[0.94660385, 0.05339615],
[0.23362354, 0.76637646],
[0.20330587, 0.79669413],
[0.333783  , 0.666217  ],
[0.61136828, 0.38863172],
[0.23513457, 0.76486543],
[0.82400676, 0.17599324],
[0.16884429, 0.83115571],
[0.93137853, 0.06862147],
[0.53440194, 0.46559806],
[0.68716648, 0.31283352],
[0.52076909, 0.47923091],
[0.73632975, 0.26367025],
[0.89867379, 0.10132621],
[0.70994113, 0.29005887],
[0.85601364, 0.14398636],
[0.09284684, 0.90715316],
[0.96629337, 0.03370663],
[0.3066029 , 0.6933971 ],
[0.86865154, 0.13134846],
[0.97029482, 0.02970518],
[0.8342464 , 0.1657536 ],
[0.4324549 , 0.5675451 ],
[0.80664318, 0.19335682],
[0.74969416, 0.25030584],
[0.18987813, 0.81012187],
[0.66750145, 0.33249855],
[0.11518964, 0.88481036],
[0.12647019, 0.87352981],
[0.88018424, 0.11981576],
[0.85975988, 0.14024012],
[0.906061  , 0.093939  ],
[0.92246813, 0.07753187],
[0.47639989, 0.52360011],
[0.1427741 , 0.8572259 ],
```

```
[0.76896699, 0.23103301],
[0.83572397, 0.16427603],
[0.06927278, 0.93072722],
[0.33457114, 0.66542886],
[0.93759669, 0.06240331],
[0.84767778, 0.15232222],
[0.76947422, 0.23052578],
[0.55218153, 0.44781847],
[0.96335471, 0.03664529],
[0.88596527, 0.11403473],
[0.85699598, 0.14300402],
[0.77839507, 0.22160493],
[0.86506761, 0.13493239],
[0.86802452, 0.13197548],
[0.93295922, 0.06704078],
[0.77638548, 0.22361452],
[0.37226787, 0.62773213],
[0.73133942, 0.26866058],
[0.77146353, 0.22853647],
[0.23802194, 0.76197806],
[0.87201793, 0.12798207],
[0.83925985, 0.16074015],
[0.70895277, 0.29104723],
[0.55420225, 0.44579775],
[0.26575849, 0.73424151],
[0.30988722, 0.69011278],
[0.34069358, 0.65930642],
[0.36265149, 0.63734851],
[0.93383866, 0.06616134],
[0.78830765, 0.21169235],
[0.95814728, 0.04185272],
[0.49761356, 0.50238644],
[0.72859611, 0.27140389],
[0.12896551, 0.87103449],
[0.69571374, 0.30428626],
[0.86062231, 0.13937769],
[0.95517846, 0.04482154],
[0.72404898, 0.27595102],
[0.8363083 , 0.1636917 ],
[0.41996575, 0.58003425],
[0.66792736, 0.33207264],
[0.83436533, 0.16563467],
[0.42751497, 0.57248503],
[0.73467252, 0.26532748],
[0.93292395, 0.06707605],
[0.45311911, 0.54688089],
[0.90916868, 0.09083132],
```

```
[0.62650398, 0.37349602],
[0.75332714, 0.24667286],
[0.98285703, 0.01714297],
[0.80157566, 0.19842434],
[0.74187644, 0.25812356],
[0.68724046, 0.31275954],
[0.7099196 , 0.2900804 ],
[0.72973092, 0.27026908],
[0.83523309, 0.16476691],
[0.36691741, 0.63308259],
[0.57148589, 0.42851411],
[0.28444974, 0.71555026],
[0.95722719, 0.04277281],
[0.96079177, 0.03920823],
[0.14725599, 0.85274401],
[0.73986931, 0.26013069],
[0.90349332, 0.09650668],
[0.77986338, 0.22013662],
[0.74419975, 0.25580025],
[0.26370149, 0.73629851],
[0.53548182, 0.46451818],
[0.25734521, 0.74265479],
[0.91548717, 0.08451283],
[0.88917167, 0.11082833],
[0.69932255, 0.30067745],
[0.60597638, 0.39402362],
[0.78584319, 0.21415681],
[0.21870737, 0.78129263],
[0.09375074, 0.90624926],
[0.58195365, 0.41804635],
[0.9475746 , 0.0524254 ],
[0.09090876, 0.90909124],
[0.80440245, 0.19559755],
[0.95809381, 0.04190619],
[0.2281093 , 0.7718907 ],
[0.86876951, 0.13123049],
[0.96019502, 0.03980498],
[0.89661361, 0.10338639],
[0.98674765, 0.01325235],
[0.64401438, 0.35598562],
[0.62612338, 0.37387662],
[0.18944878, 0.81055122],
[0.91492112, 0.08507888],
[0.91848601, 0.08151399],
[0.77267372, 0.22732628],
[0.72794938, 0.27205062],
[0.48860969, 0.51139031],
```

```
[0.6171518 , 0.3828482 ],
[0.44535515, 0.55464485],
[0.23443856, 0.76556144],
[0.74161218, 0.25838782],
[0.8793028 , 0.1206972 ],
[0.89664476, 0.10335524],
[0.18860671, 0.81139329],
[0.70937628, 0.29062372],
[0.92169631, 0.07830369],
[0.33210235, 0.66789765],
[0.10412118, 0.89587882],
[0.93956879, 0.06043121],
[0.5617854 , 0.4382146 ],
[0.04899981, 0.95100019]])
```

[46]:
```
log_pred_proba=log_model.predict_proba(x_test)[:,1]
log_pred_proba
```

[46]:
```
array([0.50281936, 0.4815264 , 0.36349778, 0.76229796, 0.31138327,
       0.28791544, 0.0612825 , 0.20556916, 0.07007615, 0.09347823,
       0.66189979, 0.14964538, 0.29774718, 0.51864567, 0.22876014,
       0.69116744, 0.18966223, 0.12025129, 0.28762688, 0.35267281,
       0.19162397, 0.10609247, 0.07830518, 0.42802892, 0.0813938 ,
       0.33449819, 0.32061317, 0.761426  , 0.21979622, 0.16496116,
       0.33414885, 0.64943303, 0.12855523, 0.87808414, 0.5345092 ,
       0.24042728, 0.88034704, 0.39009513, 0.31393857, 0.10677779,
       0.48053339, 0.07832231, 0.66648425, 0.33452695, 0.01745494,
       0.1544139 , 0.22369263, 0.08894943, 0.15671555, 0.05339615,
       0.76637646, 0.79669413, 0.666217  , 0.38863172, 0.76486543,
       0.17599324, 0.83115571, 0.06862147, 0.46559806, 0.31283352,
       0.47923091, 0.26367025, 0.10132621, 0.29005887, 0.14398636,
       0.90715316, 0.03370663, 0.6933971 , 0.13134846, 0.02970518,
       0.1657536 , 0.5675451 , 0.19335682, 0.25030584, 0.81012187,
       0.33249855, 0.88481036, 0.87352981, 0.11981576, 0.14024012,
       0.093939  , 0.07753187, 0.52360011, 0.8572259 , 0.23103301,
       0.16427603, 0.93072722, 0.66542886, 0.06240331, 0.15232222,
       0.23052578, 0.44781847, 0.03664529, 0.11403473, 0.14300402,
       0.22160493, 0.13493239, 0.13197548, 0.06704078, 0.22361452,
       0.62773213, 0.26866058, 0.22853647, 0.76197806, 0.12798207,
       0.16074015, 0.29104723, 0.44579775, 0.73424151, 0.69011278,
       0.65930642, 0.63734851, 0.06616134, 0.21169235, 0.04185272,
       0.50238644, 0.27140389, 0.87103449, 0.30428626, 0.13937769,
       0.04482154, 0.27595102, 0.1636917 , 0.58003425, 0.33207264,
       0.16563467, 0.57248503, 0.26532748, 0.06707605, 0.54688089,
       0.09083132, 0.37349602, 0.24667286, 0.01714297, 0.19842434,
       0.25812356, 0.31275954, 0.2900804 , 0.27026908, 0.16476691,
       0.63308259, 0.42851411, 0.71555026, 0.04277281, 0.03920823,
```

```
       0.85274401, 0.26013069, 0.09650668, 0.22013662, 0.25580025,
       0.73629851, 0.46451818, 0.74265479, 0.08451283, 0.11082833,
       0.30067745, 0.39402362, 0.21415681, 0.78129263, 0.90624926,
       0.41804635, 0.0524254 , 0.90909124, 0.19559755, 0.04190619,
       0.7718907 , 0.13123049, 0.03980498, 0.10338639, 0.01325235,
       0.35598562, 0.37387662, 0.81055122, 0.08507888, 0.08151399,
       0.22732628, 0.27205062, 0.51139031, 0.3828482 , 0.55464485,
       0.76556144, 0.25838782, 0.1206972 , 0.10335524, 0.81139329,
       0.29062372, 0.07830369, 0.66789765, 0.89587882, 0.06043121,
       0.4382146 , 0.95100019])
```

[47]:

```
fpr,tpr,thresholds=roc_curve(y_test,log_pred_proba)
print(fpr)
print("\n")
print(tpr)
print("\n")
print(thresholds)
```

```
[0.         0.         0.00769231 0.00769231 0.03076923 0.03076923
 0.03846154 0.03846154 0.05384615 0.05384615 0.06153846 0.06153846
 0.08461538 0.08461538 0.1        0.1        0.10769231 0.10769231
 0.11538462 0.11538462 0.16923077 0.16923077 0.17692308 0.17692308
 0.20769231 0.20769231 0.23076923 0.23076923 0.25384615 0.25384615
 0.28461538 0.28461538 0.29230769 0.29230769 0.36923077 0.36923077
 0.56923077 0.56923077 0.57692308 0.57692308 0.6        0.6
 0.62307692 0.62307692 0.63846154 0.63846154 0.66153846 0.66153846
 0.88461538 0.88461538 1.        ]


[0.         0.01612903 0.01612903 0.20967742 0.20967742 0.32258065
 0.32258065 0.43548387 0.43548387 0.48387097 0.48387097 0.51612903
 0.51612903 0.56451613 0.56451613 0.58064516 0.58064516 0.61290323
 0.61290323 0.66129032 0.66129032 0.69354839 0.69354839 0.72580645
 0.72580645 0.74193548 0.74193548 0.79032258 0.79032258 0.80645161
 0.80645161 0.82258065 0.82258065 0.85483871 0.85483871 0.87096774
 0.87096774 0.88709677 0.88709677 0.91935484 0.91935484 0.93548387
 0.93548387 0.9516129  0.9516129  0.96774194 0.96774194 0.98387097
 0.98387097 1.         1.        ]


[1.95100019 0.95100019 0.93072722 0.83115571 0.81012187 0.76229796
 0.76197806 0.69116744 0.66789765 0.66542886 0.66189979 0.64943303
 0.62773213 0.5675451  0.54688089 0.5345092  0.52360011 0.51139031
 0.50281936 0.48053339 0.42851411 0.41804635 0.39402362 0.38863172
 0.36349778 0.35598562 0.33449819 0.33207264 0.31283352 0.31275954
 0.29774718 0.29104723 0.29062372 0.29005887 0.26013069 0.25838782
 0.16563467 0.16496116 0.16476691 0.1636917  0.1544139  0.15232222
```

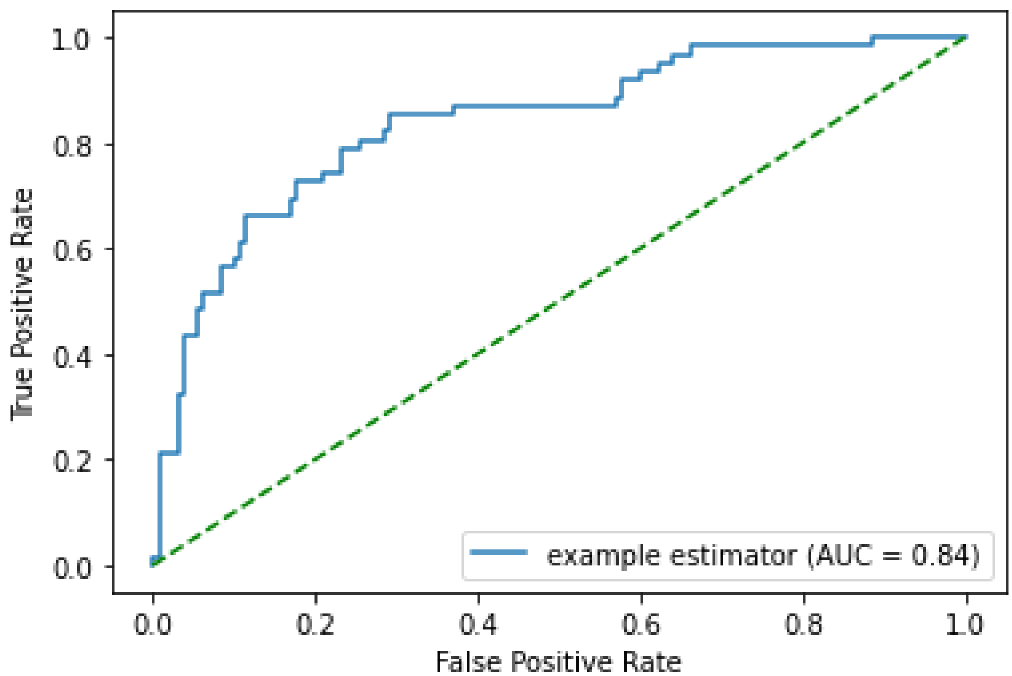
 0.14300402 0.14024012 0.13493239 0.13197548 0.12855523 0.12798207
 0.06240331 0.0612825  0.01325235]

[48]:
```
roc_auc=auc(fpr,tpr)
roc_auc
```

[48]: 0.8388337468982631

[49]:
```
display=RocCurveDisplay(fpr=fpr,tpr=tpr,roc_auc=roc_auc,estimator_name="example␣
↪estimator")
display.plot()
plt.plot(fpr,fpr,"r--",color="green")
```

[49]: [<matplotlib.lines.Line2D at 0x1ba8fa7bbe0>]




### 1.10 Support vector machine

[50]:
```
from sklearn.svm import SVC
```

[51]:
```
svc_model=SVC(probability=True)
svc_model.fit(x_train,y_train)
svc_pred=svc_model.predict(x_test)
svc_pred
```

[51]:
```
array([0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1], dtype=int64)
```

[52]:
```python
accuracy_score(svc_pred,y_test)
```

[52]:
```
0.78125
```

[53]:
```python
confusion_matrix(svc_pred,y_test)
```

[53]:
```
array([[116,  28],
       [ 14,  34]], dtype=int64)
```

[54]:
```python
print("Sensitivity is ",116/(116+34))
```

```
Sensitivity is  0.7733333333333333
```

[55]:
```python
print("Specificity is ",25/(28+14))
```

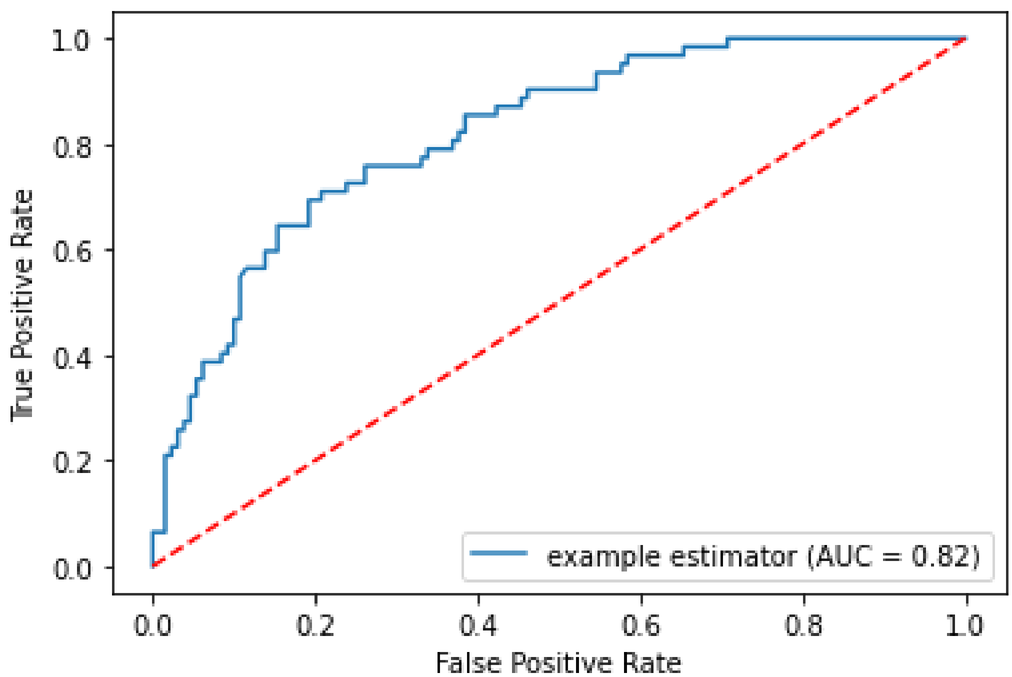
```
Specificity is  0.5952380952380952
```

[56]:
```python
print(classification_report(svc_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       0.89      0.81      0.85       144
           1       0.55      0.71      0.62        48

    accuracy                           0.78       192
   macro avg       0.72      0.76      0.73       192
weighted avg       0.81      0.78      0.79       192
```

[57]:
```python
svc_pred_proba=svc_model.predict_proba(x_test)[:,1]
fpr,tpr,thresholds=roc_curve(y_test,svc_pred_proba)
roc_auc=auc(fpr,tpr)
display=RocCurveDisplay(fpr=fpr,tpr=tpr,roc_auc=roc_auc,estimator_name="example estimator")
display.plot()
plt.plot(fpr,fpr,"r--",color="red")
```

[57]:
```
[<matplotlib.lines.Line2D at 0x1ba8fad6a90>]
```




### 1.11 Decision tree

[58]:
```
from sklearn.tree import DecisionTreeClassifier
dt_model=DecisionTreeClassifier(max_depth=5)
dt_model.fit(x_train,y_train)
dt_pred=dt_model.predict(x_test)
dt_pred
```

[58]:
```
array([0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0,
       0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1,
       0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1,
       0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0,
       1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0], dtype=int64)
```

[59]:
```
accuracy_score(dt_pred,y_test)
```

[59]: 0.7291666666666666

[60]:
```
confusion_matrix(dt_pred,y_test)
```

[60]:
```
array([[100,  22],
       [ 30,  40]], dtype=int64)
```

[61]:
```
print("Sensitivity : ",100/(100+40))
```

```
Sensitivity :  0.7142857142857143
```

[62]:
```
print("Specificity : ",22+(22+30))
```

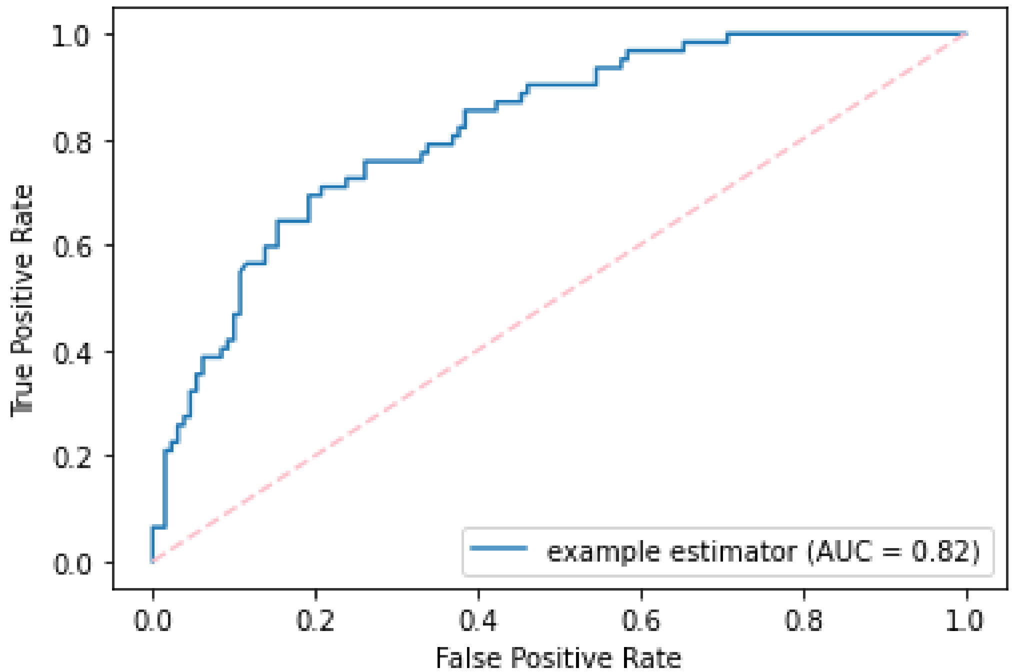
```
Specificity :  74
```

[63]:
```
print(classification_report(dt_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       0.77      0.82      0.79       122
           1       0.65      0.57      0.61        70

    accuracy                           0.73       192
   macro avg       0.71      0.70      0.70       192
weighted avg       0.72      0.73      0.73       192
```

[64]:
```
dt_pred_proba=dt_model.predict_proba(x_test)[:,1]
fpr,tpr,thresholds=roc_curve(y_test,svc_pred_proba)
roc_auc=auc(fpr,tpr)
display=RocCurveDisplay(fpr=fpr,tpr=tpr,roc_auc=roc_auc,estimator_name="example␣
↪estimator")
display.plot()
plt.plot(fpr,fpr,"r--",color="pink")
```

[64]:
```
[<matplotlib.lines.Line2D at 0x1ba8fcb4850>]
```




### 1.12 Random forest

[65]:
```
from sklearn.ensemble import RandomForestClassifier
rf_model=RandomForestClassifier(n_estimators=25)
rf_model.fit(x_train,y_train)
rf_pred=rf_model.predict(x_test)
rf_pred
```

[65]:
```
array([0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1,
       1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0,
       0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1], dtype=int64)
```

[66]:
```
accuracy_score(rf_pred,y_test)
```

[66]: 0.7708333333333334

[67]:
```
confusion_matrix(rf_pred,y_test)
```

[67]:
```
array([[109,  23],
       [ 21,  39]], dtype=int64)
```

[68]:
```
print("Sensitivity :",113/(113+41))
```

```
Sensitivity : 0.7337662337662337
```

[69]:
```
print("Specificity :",21/(21+17))
```

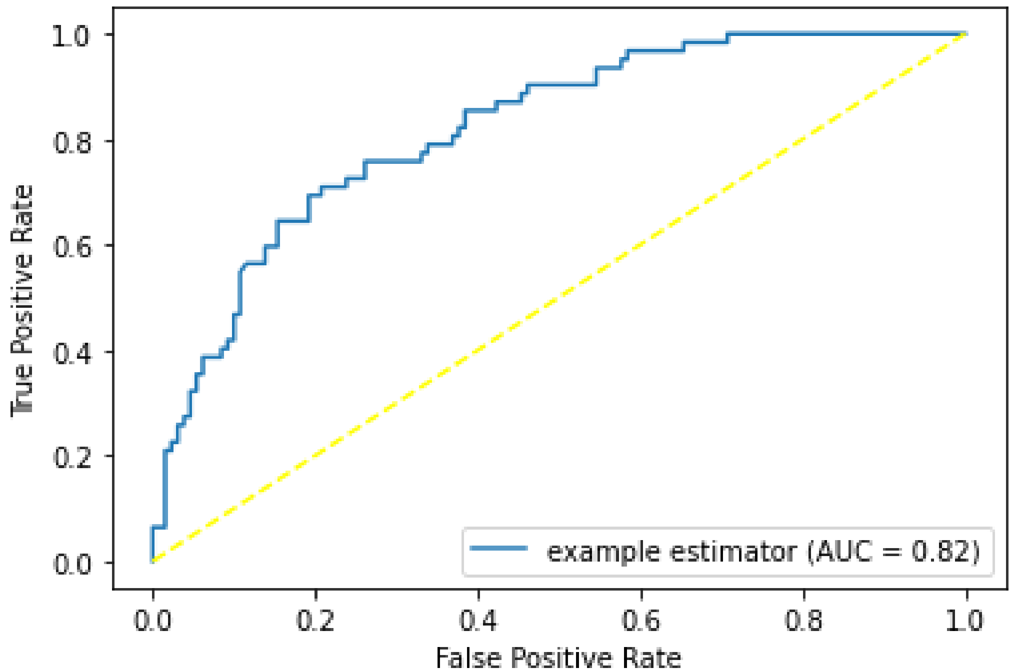
```
Specificity : 0.5526315789473685
```

[70]:
```
print(classification_report(rf_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       0.84      0.83      0.83       132
           1       0.63      0.65      0.64        60

    accuracy                           0.77       192
   macro avg       0.73      0.74      0.74       192
weighted avg       0.77      0.77      0.77       192
```

[71]:
```
rf_pred_proba=rf_model.predict_proba(x_test)[:,1]
fpr,tpr,thresholds=roc_curve(y_test,svc_pred_proba)
roc_auc=auc(fpr,tpr)
display=RocCurveDisplay(fpr=fpr,tpr=tpr,roc_auc=roc_auc,estimator_name="example␣
↪estimator")
display.plot()
plt.plot(fpr,fpr,"r--",color="yellow")
```

[71]:
```
[<matplotlib.lines.Line2D at 0x1ba8fd8d8b0>]
```




### 1.13 K nearest neighbors

[72]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_model=KNeighborsClassifier(n_neighbors=5)
knn_model.fit(x_train,y_train)
knn_pred=knn_model.predict(x_test)
knn_pred
```

[72]:
```
array([0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1], dtype=int64)
```

[73]:
```
accuracy_score(knn_pred,y_test)
```

[73]: 0.7447916666666666

[74]:
```
confusion_matrix(knn_pred,y_test)
```

[74]:
```
array([[110,  29],
       [ 20,  33]], dtype=int64)
```

[75]:
```
print("Sensitivity: ",110/(110+33))
```

```
Sensitivity:  0.7692307692307693
```

[76]:
```
print("Specificity :",29/(29+20))
```

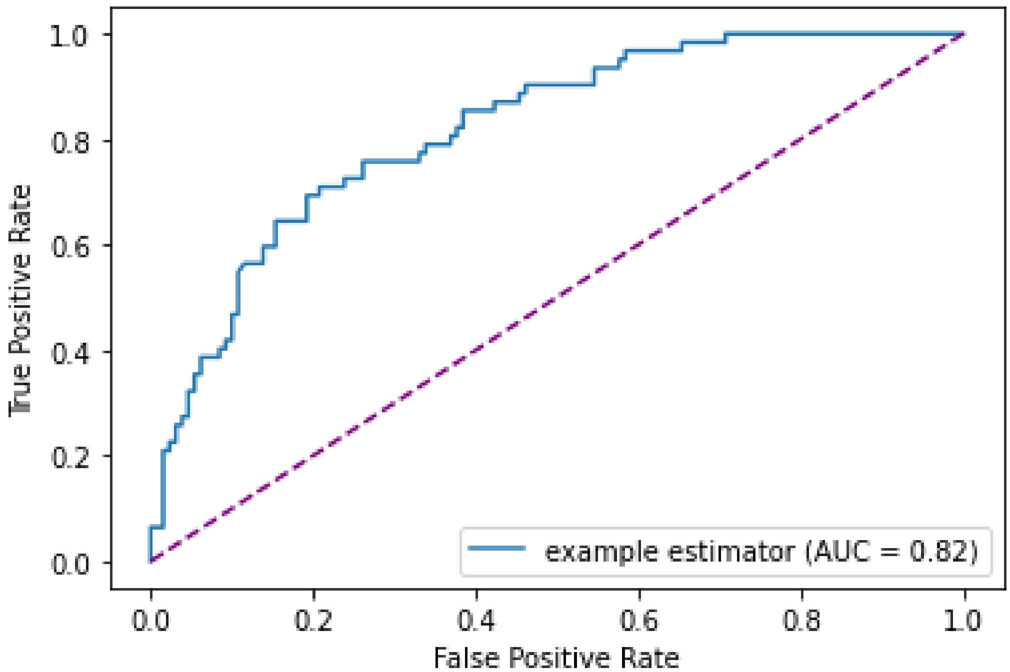
```
Specificity : 0.5918367346938775
```

[77]:
```
print(classification_report(knn_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       0.85      0.79      0.82       139
           1       0.53      0.62      0.57        53

    accuracy                           0.74       192
   macro avg       0.69      0.71      0.70       192
weighted avg       0.76      0.74      0.75       192
```

[78]:
```
knn_pred_proba=knn_model.predict_proba(x_test)[:,1]
fpr,tpr,thresholds=roc_curve(y_test,svc_pred_proba)
roc_auc=auc(fpr,tpr)
display=RocCurveDisplay(fpr=fpr,tpr=tpr,roc_auc=roc_auc,estimator_name="example␣
↪estimator")
display.plot()
plt.plot(fpr,fpr,"r--",color="purple")
```

[78]:
```
[<matplotlib.lines.Line2D at 0x1ba8fdf8820>]
```




#### 1.13.1 From the above models,it is clear that Logistic regresssion and Random forest are best models for this dataset.

#### 1.13.2 THANK YOU...!!!