diabetes

May 15, 2024

IMPORTING LIBRARIES

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
df=pd.read_csv(r'C:\Users\ADMIN\Desktop\diabetes.csv')
```

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

[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

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

[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

```
     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
```

[5]:
```
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
```

[6]:
```
df.describe()
```

[6]:
```
       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
```

CHECKING FOR MISSING VALUES

[7]:
```
df.isnull().sum()
```

[7]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

[8]:
```
df.groupby('Pregnancies')[['BloodPressure','Age','Outcome']].value_counts()
```

[8]:
```
Pregnancies  BloodPressure  Age  Outcome
0            64             21   0          4
             76             26   0          2
             80             27   0          2
             64             22   0          2
             68             21   0          2
                                            ..
13           88             39   0          1
14           78             46   1          1
             62             38   1          1
15           70             43   1          1
17           72             47   1          1
Name: count, Length: 704, dtype: int64
```

[9]:
```
df.groupby('Age')[['DiabetesPedigreeFunction','Outcome']].value_counts()
```

[9]:
```
Age  DiabetesPedigreeFunction  Outcome
21   0.559                     0          2
     0.289                     0          2
     0.299                     0          2
     0.148                     0          2
     0.078                     0          1
                                           ..
69   0.186                     0          1
     0.640                     0          1
70   0.235                     1          1
72   0.832                     0          1
81   0.460                     0          1
Name: count, Length: 759, dtype: int64
```

[10]:
```
df[df['Age']==20].sum()
```

[10]:
```
Pregnancies                 0.0
Glucose                     0.0
BloodPressure               0.0
SkinThickness               0.0
Insulin                     0.0
BMI                         0.0
DiabetesPedigreeFunction    0.0
Age                         0.0
Outcome                     0.0
dtype: float64
```

[11]:
```
df[df['Pregnancies']==10]
```

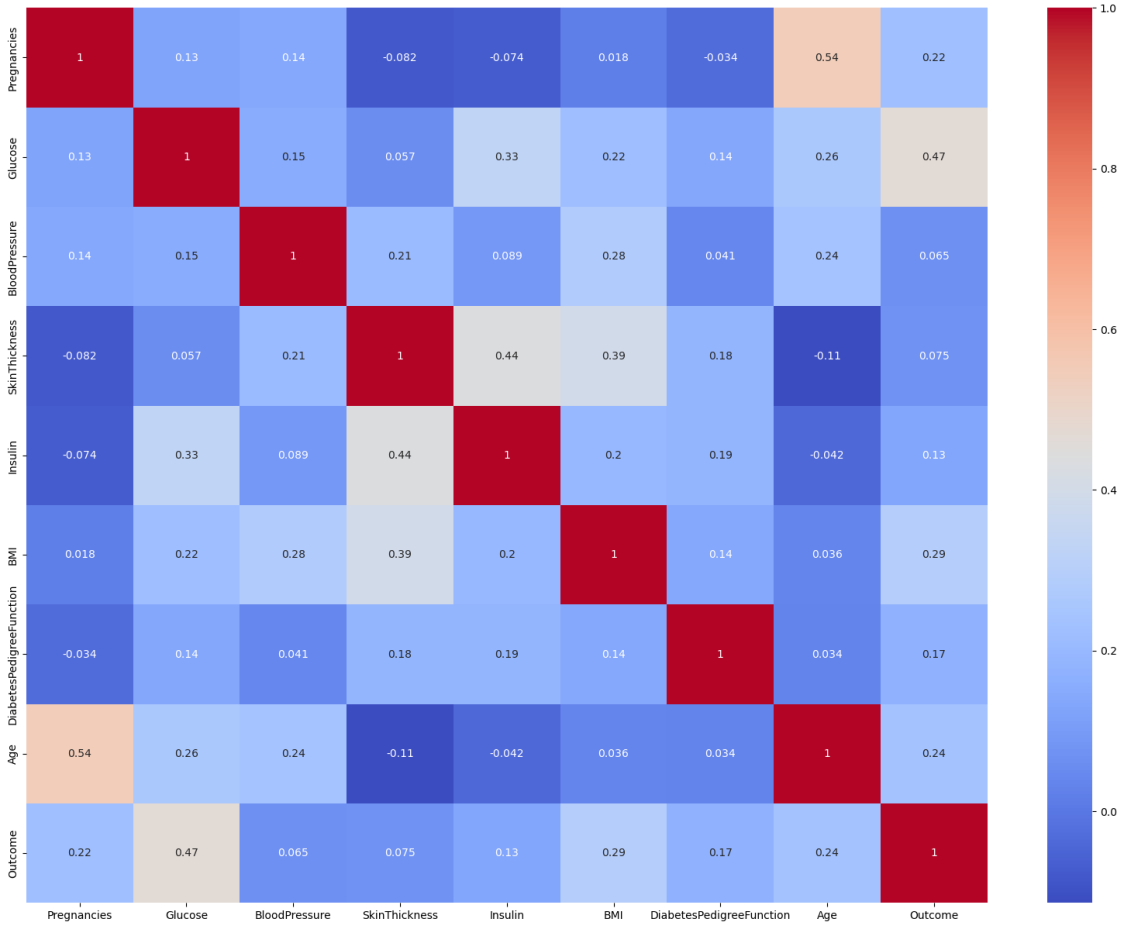
[11]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
7	10	115	0	0	0	35.3	0.134	29	0
11	10	168	74	0	0	38.0	0.537	34	1
12	10	139	80	0	0	27.1	1.441	57	0
25	10	125	70	26	115	31.1	0.205	41	1
34	10	122	78	31	0	27.6	0.512	45	0
143	10	108	66	0	0	32.4	0.272	42	1
246	10	122	68	0	0	31.2	0.258	41	0
270	10	101	86	37	0	45.6	1.136	38	1
281	10	129	76	28	122	35.9	0.280	39	0
306	10	161	68	23	132	25.5	0.326	47	1
327	10	179	70	0	0	35.1	0.200	37	0
458	10	148	84	48	237	37.6	1.001	51	1
464	10	115	98	0	0	24.0	1.022	34	0
505	10	75	82	0	0	33.3	0.263	38	0
542	10	90	85	32	0	34.9	0.825	56	1
578	10	133	68	0	0	27.0	0.245	36	0
634	10	92	62	0	0	25.9	0.167	31	0
660	10	162	84	0	0	27.7	0.182	54	0
667	10	111	70	27	0	27.5	0.141	40	1
672	10	68	106	23	49	35.5	0.285	47	0
706	10	115	0	0	0	0.0	0.261	30	1
712	10	129	62	36	0	41.2	0.441	38	1
717	10	94	72	18	0	23.1	0.595	56	0
763	10	101	76	48	180	32.9	0.171	63	0

270	1.136	38	1
281	0.280	39	0
306	0.326	47	1
327	0.200	37	0
458	1.001	51	1
464	1.022	34	0
505	0.263	38	0
542	0.825	56	1
578	0.245	36	0
634	0.167	31	0
660	0.182	54	0
667	0.141	40	1
672	0.285	47	0
706	0.261	30	1
712	0.441	38	1
717	0.595	56	0
763	0.171	63	0

CHECKING CORRELATION OF THE DATASET

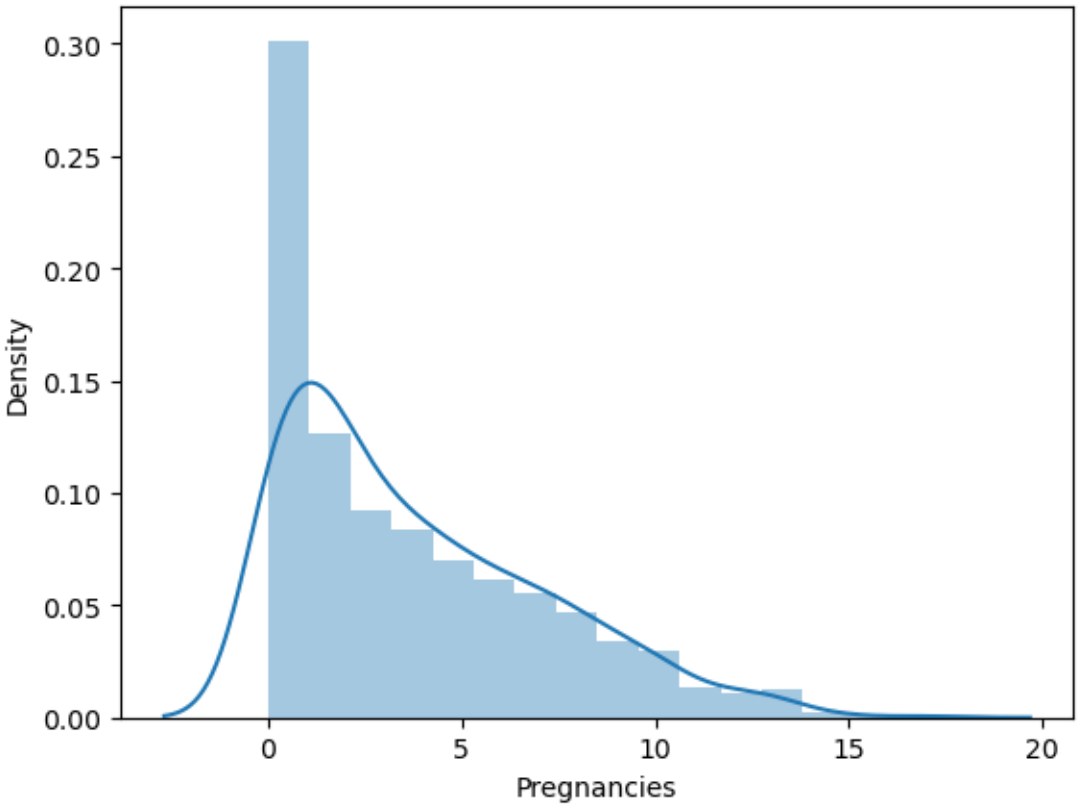
[12]:
```
plt.figure(figsize=(20,15))
sns.heatmap(df.corr(),annot=True,cmap='coolwarm')
plt.show()
```

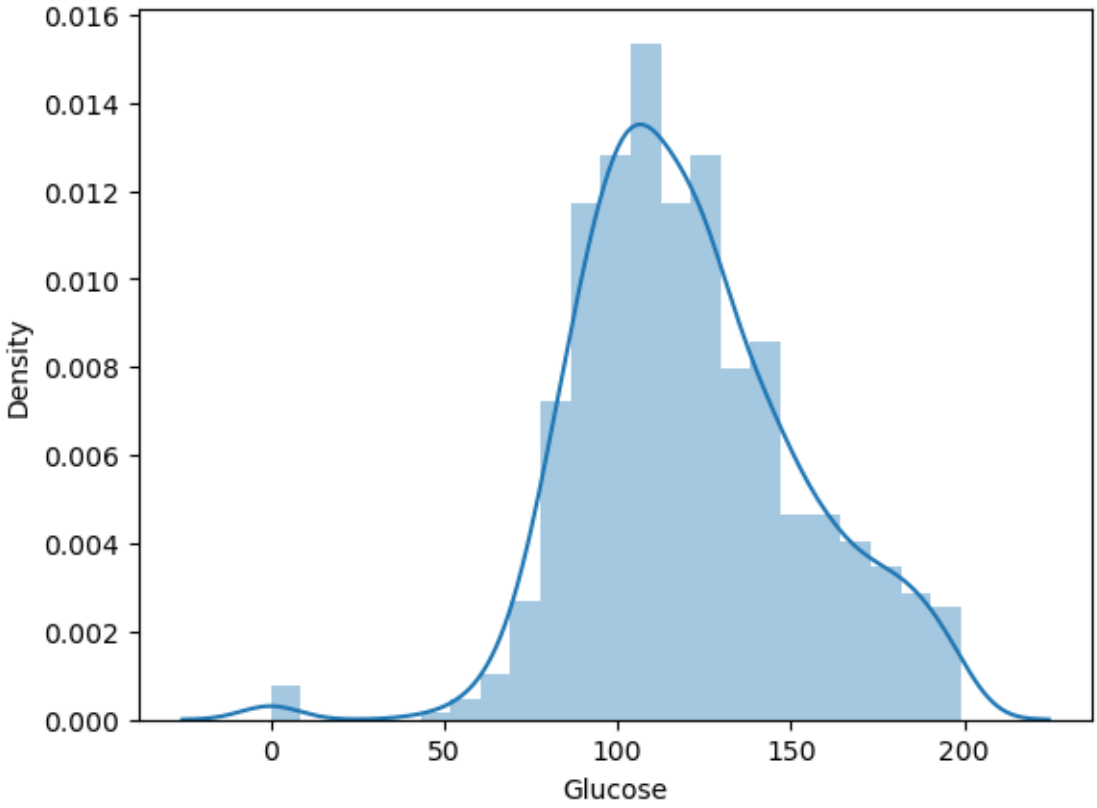


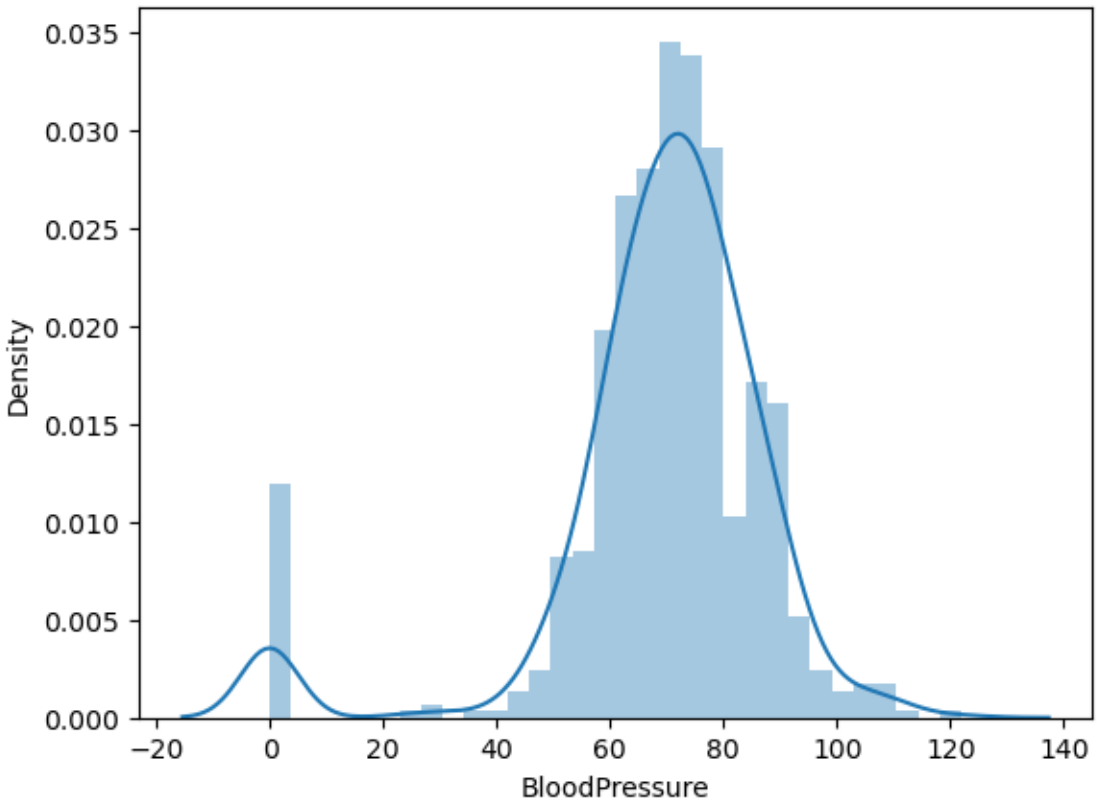
DATA VISUALIZATION

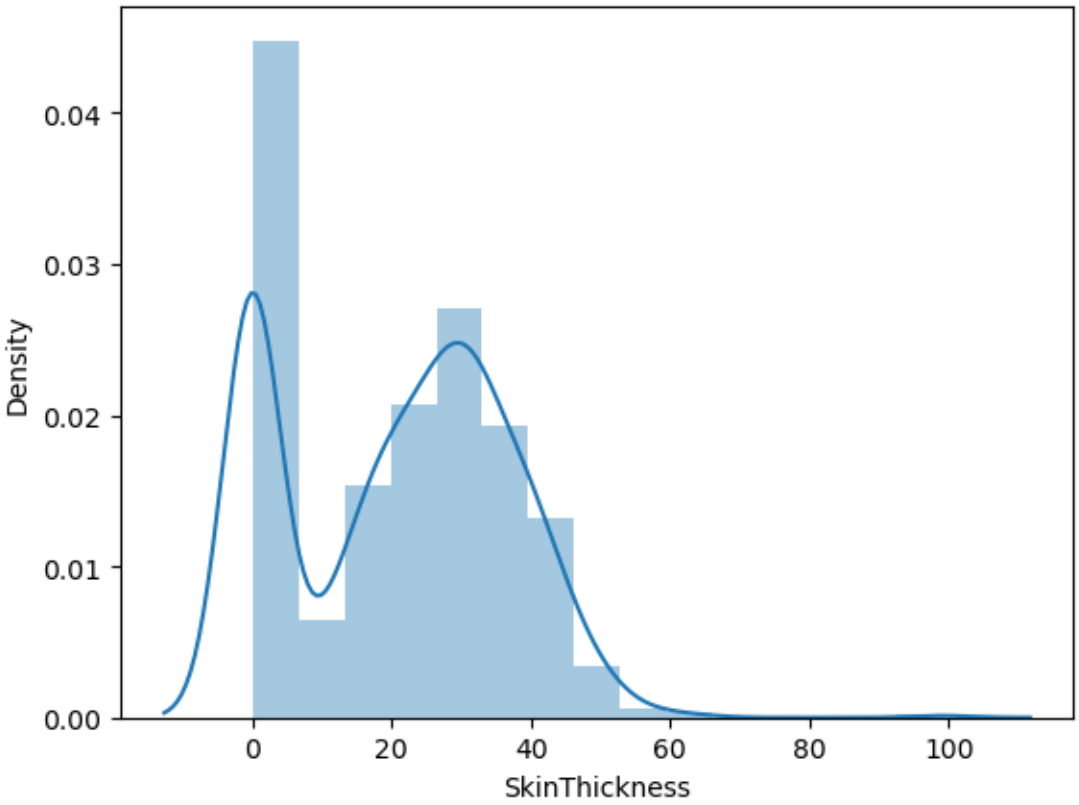
[13]:
```
def distplots(col):
    sns.distplot(df[col])
    plt.show()
```

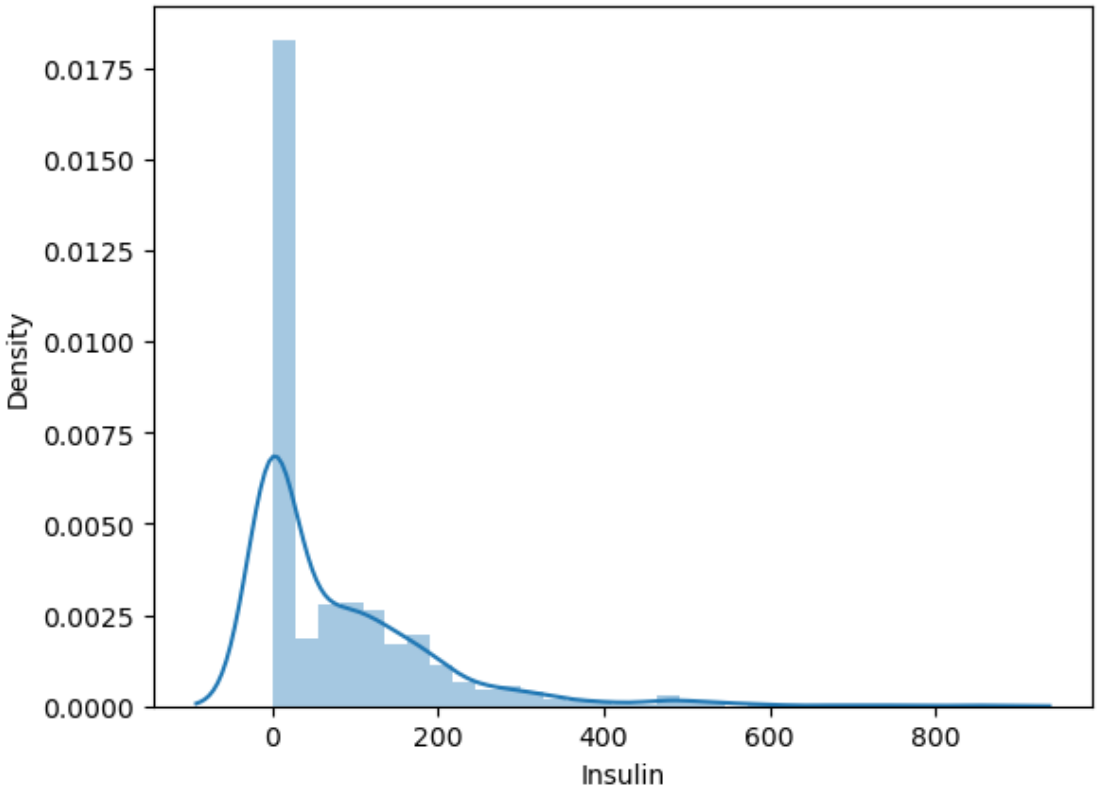
[14]:
```
for i in list(df.select_dtypes(exclude=['object']).columns):
    distplots(i)
```

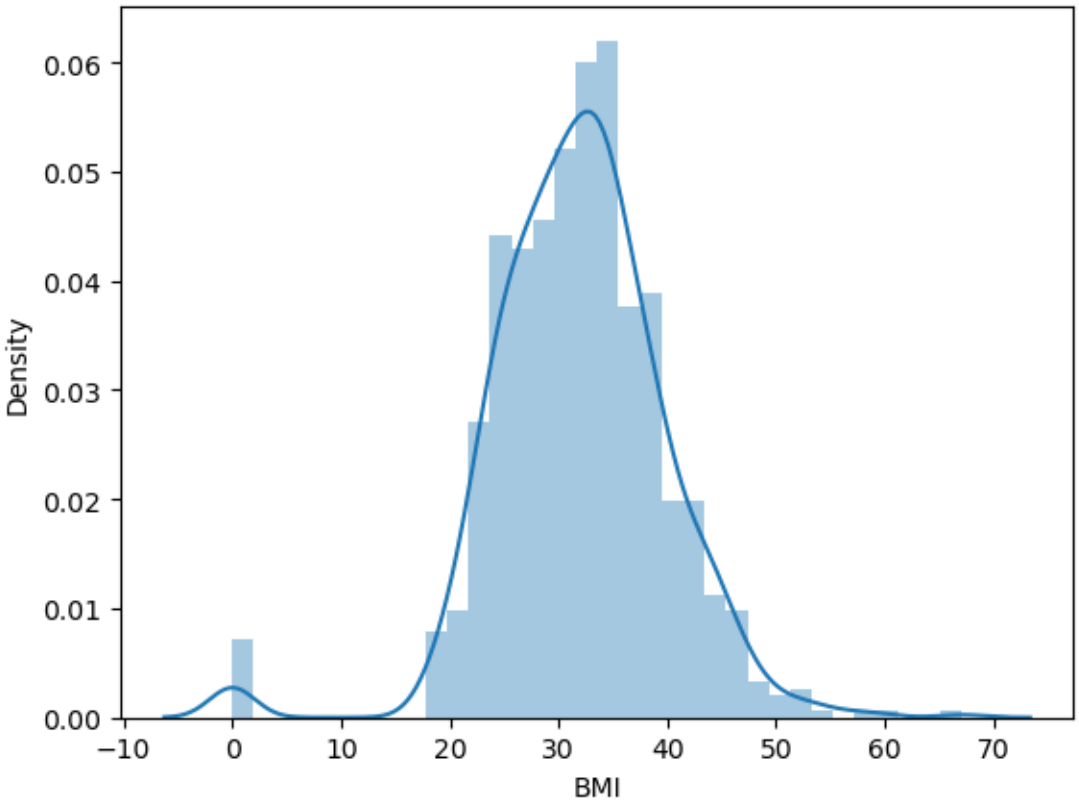

0.30
0.25
0.20
0.15
0.10
0.05
0.00
Density
0
5
10
15
20
Pregnancies

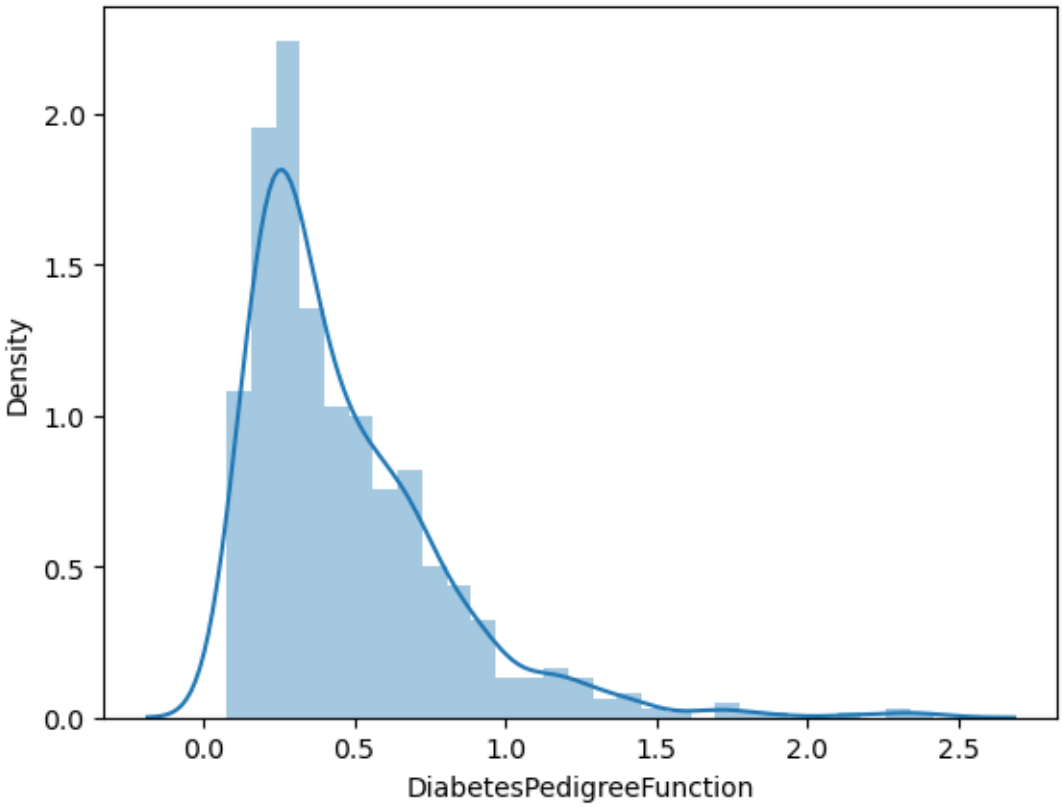

0.016
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
Density
0
50
100
150
200
Glucose

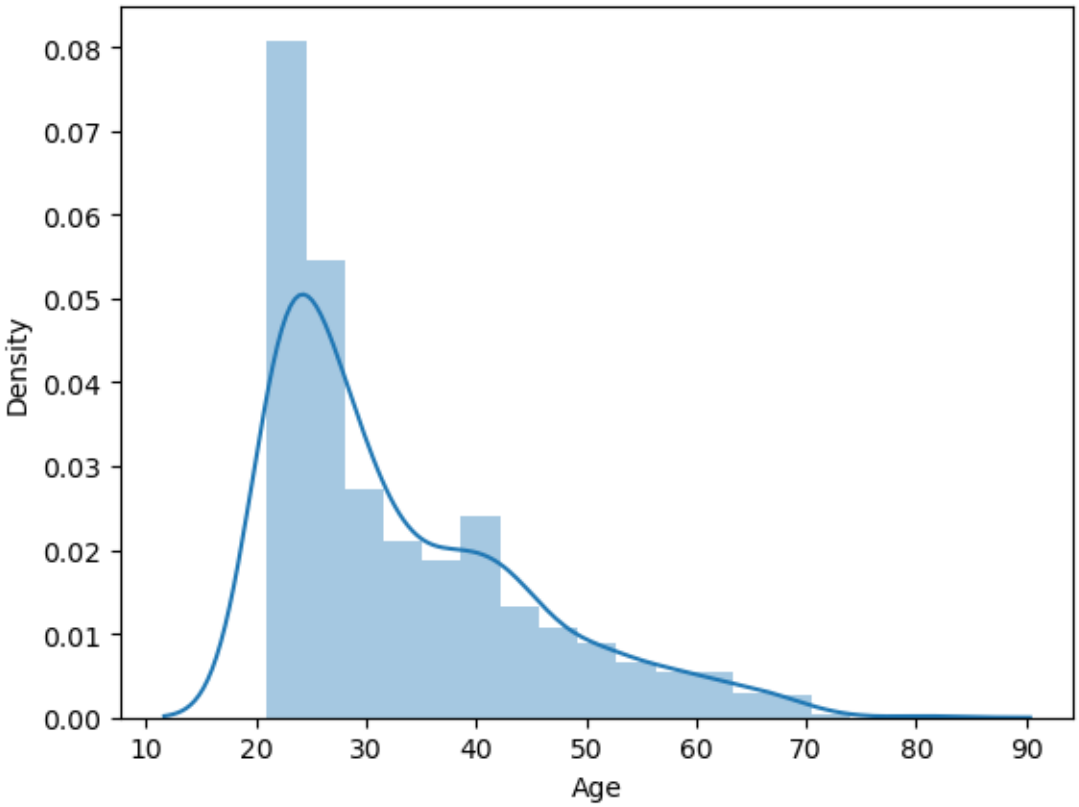

0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
Density
−20
0
20
40
60
80
100
120
140
BloodPressure

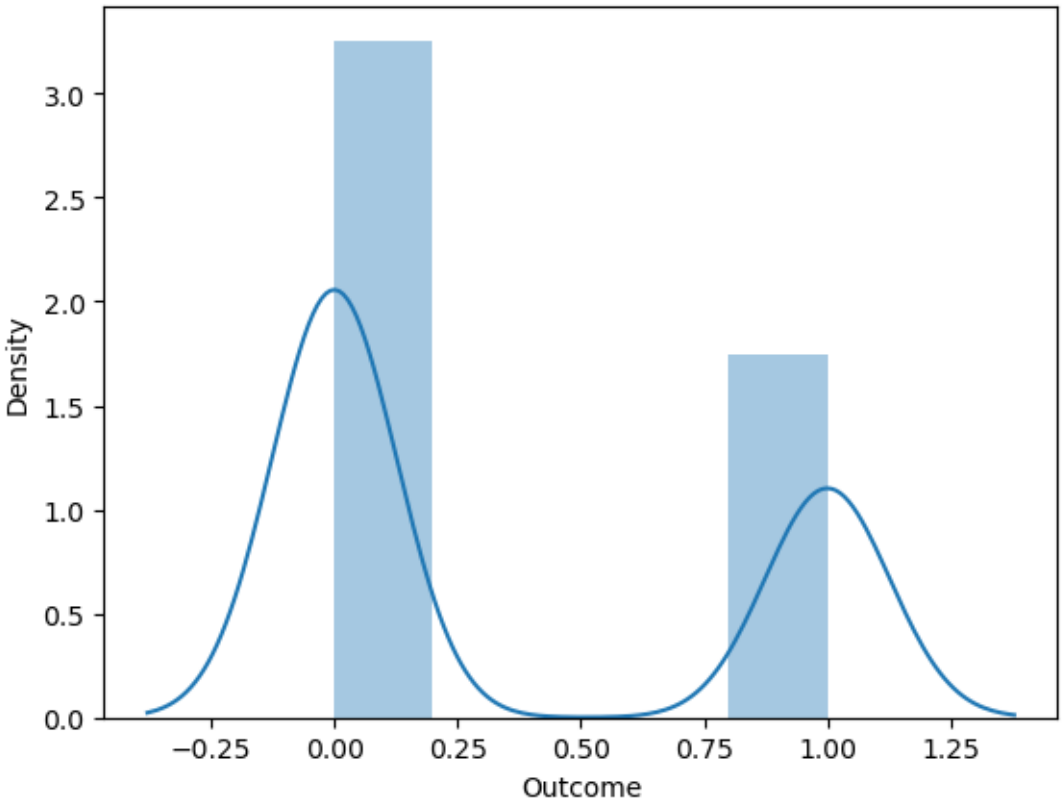

0.04
0.03
0.02
0.01
0.00
Density
0
20
40
60
80
100
SkinThickness


0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
Density
0
200
400
600
800
Insulin


0.06
0.05
0.04
0.03
0.02
0.01
0.00
Density
−10
0
10
20
30
40
50
60
70
BMI


Density
2.0
1.5
1.0
0.5
0.0
0.0
0.5
1.0
1.5
2.0
2.5
DiabetesPedigreeFunction


0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
Density
10
20
30
40
50
60
70
80
90
Age



Split the data into independent and dependent variable

[15]:
```
x=df.drop(['Outcome'],axis=1)
y=df['Outcome']
```

FEATURE SCALING

[16]:
```
from sklearn.preprocessing import StandardScaler
scala=StandardScaler()
x=scala.fit_transform(x)
```

[17]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=2)
```

LOGISTIC REGRESSION

[19]:
```
from sklearn.linear_model import LogisticRegression
lreg=LogisticRegression()
lreg.fit(x_train,y_train)
```

[19]:
```
LogisticRegression()
```

[20]:
```
from sklearn.metrics import␣
↪classification_report,confusion_matrix,accuracy_score
```

[21]:
```
yhat_train_lreg=lreg.predict(x_train)
yhat_test_lreg=lreg.predict(x_test)
```

EVALUATION OF MATRIX

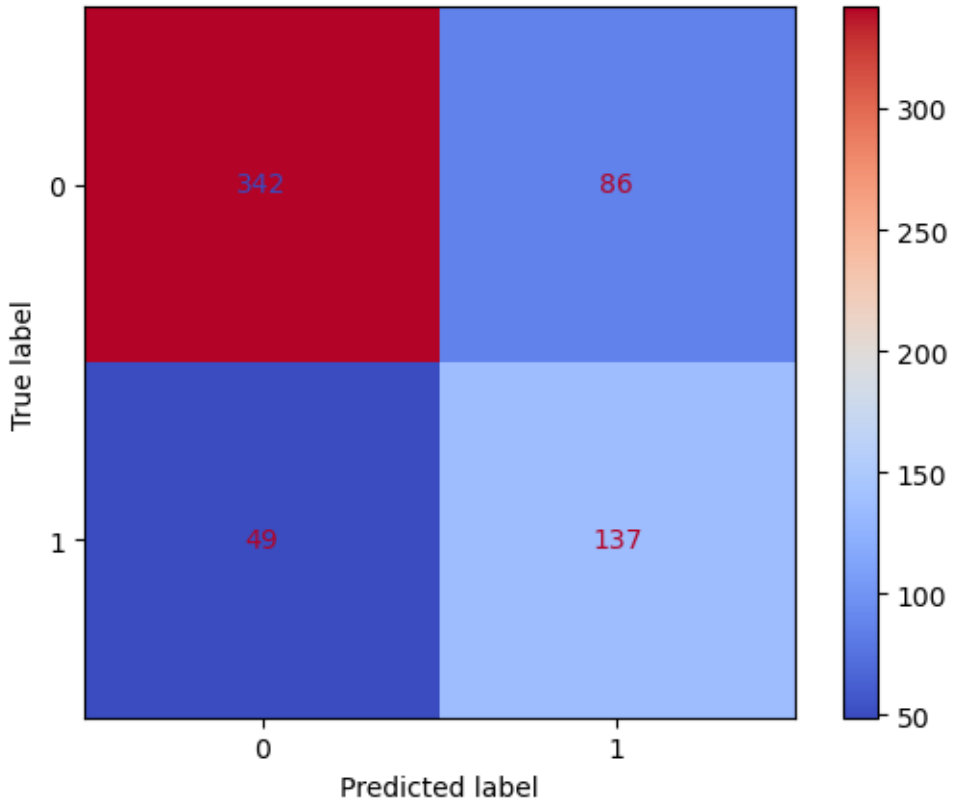
[22]:
```
print(classification_report(y_train,yhat_train_lreg))
print()
print(classification_report(y_test,yhat_test_lreg))
```

```
              precision    recall  f1-score   support

           0       0.80      0.87      0.84       391
           1       0.74      0.61      0.67       223

    accuracy                           0.78       614
   macro avg       0.77      0.74      0.75       614
weighted avg       0.78      0.78      0.78       614


              precision    recall  f1-score   support

           0       0.81      0.88      0.84       109
           1       0.63      0.49      0.55        45

    accuracy                           0.77       154
   macro avg       0.72      0.68      0.70       154
weighted avg       0.75      0.77      0.76       154
```

[23]:
```
print(confusion_matrix(y_train,yhat_train_lreg))
print()
print(confusion_matrix(y_test,yhat_test_lreg))
```

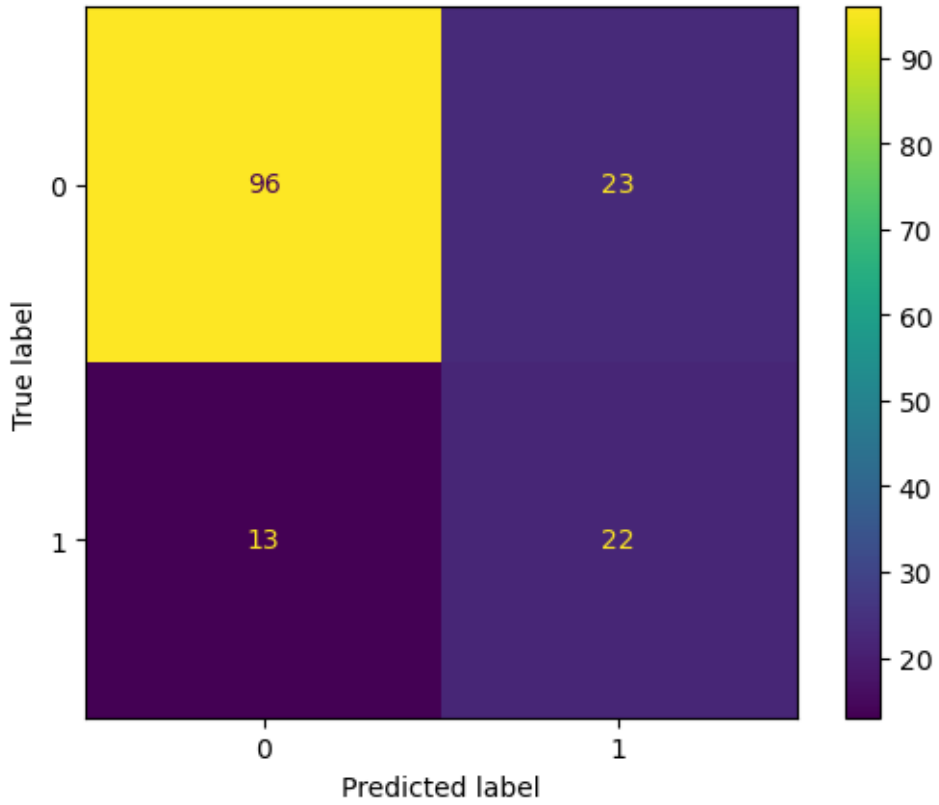
```
[[342  49]
 [ 86 137]]

[[96 13]
 [23 22]]
```

[24]:
```
from sklearn.metrics import ConfusionMatrixDisplay
c_matrix=ConfusionMatrixDisplay(confusion_matrix(yhat_train_lreg,y_train))
c_matrix.plot(cmap='coolwarm')
plt.show()
```



[25]:
```
c_matrix=ConfusionMatrixDisplay(confusion_matrix(yhat_test_lreg,y_test))
c_matrix.plot(cmap=plt.cm.viridis)
```

[25]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a0c7958c90>



[26]:
```
from sklearn.metrics import accuracy_score
print('Accuracy: ',accuracy_score(y_train,yhat_train_lreg))
print()
print('Accuracy: ', accuracy_score(y_test,yhat_test_lreg))
```

```
Accuracy:  0.7801302931596091

Accuracy:  0.7662337662337663
```

[18]:
```
from sklearn.model_selection import cross_val_score
```

[27]:
```
scores=cross_val_score(lreg,x,y,cv=3)
np.mean(scores)
```

[27]: 0.7708333333333334

SUPPORT VENDOR MACHINE

[28]:
```
from sklearn import svm
s2=svm.SVC(kernel='linear')
s2.fit(x_train,y_train)
```

[28]:
```
SVC(kernel='linear')
```

[29]:
```
yhat1_train_s2=s2.predict(x_train)
yhat1_test_s2=s2.predict(x_test)
```

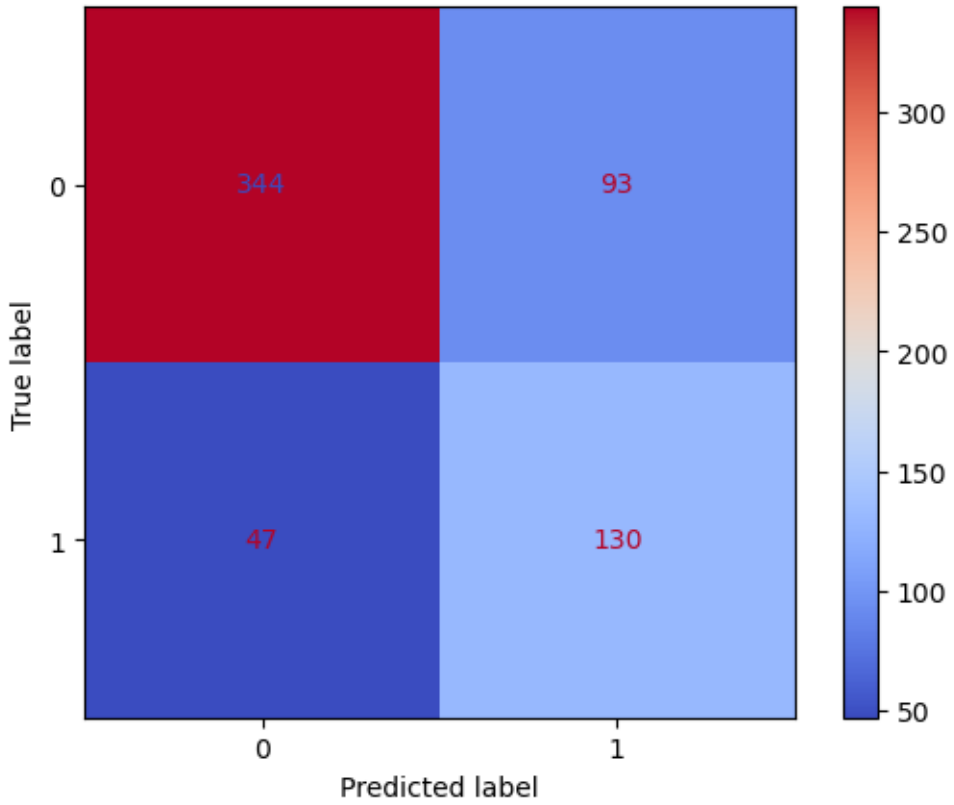
[30]:
```
print(classification_report(y_train,yhat1_train_s2))
print()
print(classification_report(y_test,yhat1_test_s2))
```

```
              precision    recall  f1-score   support

           0       0.79      0.88      0.83       391
           1       0.73      0.58      0.65       223

    accuracy                           0.77       614
   macro avg       0.76      0.73      0.74       614
weighted avg       0.77      0.77      0.77       614


              precision    recall  f1-score   support

           0       0.81      0.88      0.84       109
           1       0.63      0.49      0.55        45

    accuracy                           0.77       154
   macro avg       0.72      0.68      0.70       154
weighted avg       0.75      0.77      0.76       154
```

[31]:
```
print(confusion_matrix(y_train,yhat1_train_s2))
print()
print(confusion_matrix(y_test,yhat1_test_s2))
```

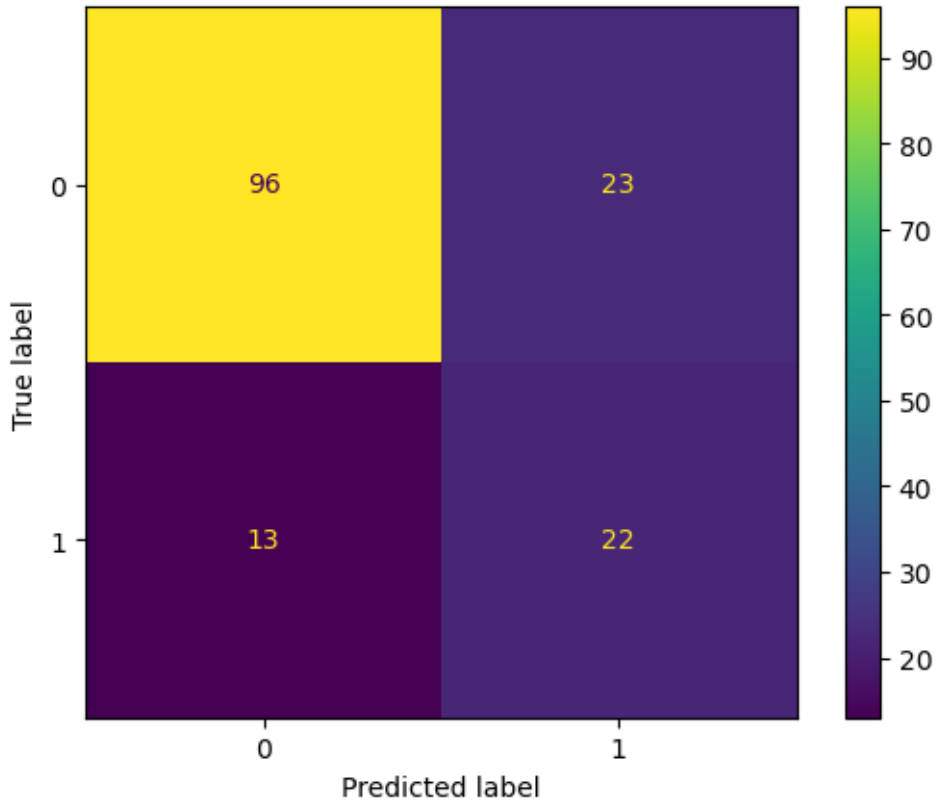
```
[[344  47]
 [ 93 130]]

[[96 13]
 [23 22]]
```

[32]:
```
c_matrix=ConfusionMatrixDisplay(confusion_matrix(yhat1_train_s2,y_train))
c_matrix.plot(cmap='coolwarm')
plt.show()
```



[33]:
```
c_matrix=ConfusionMatrixDisplay(confusion_matrix(yhat1_test_s2,y_test))
c_matrix.plot(cmap=plt.cm.viridis)
```

[33]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a0c79c75d0>



[34]:
```
from sklearn.metrics import accuracy_score
print('Accuracy: ',accuracy_score(y_train,yhat1_train_s2))
print()
print('Accuracy: ', accuracy_score(y_test,yhat1_test_s2))
```

```
Accuracy:  0.7719869706840391

Accuracy:  0.7662337662337663
```

[35]:
```
scores=cross_val_score(s2,x,y,cv=3)
np.mean(scores)
```

[35]:
```
0.7591145833333334
```

RANDOM FOREST

[36]:
```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

[36]:
```
RandomForestClassifier()
```

[37]:
```
yhat2_train_rfc=rfc.predict(x_train)
yhat2_test_rfc=rfc.predict(x_test)
```

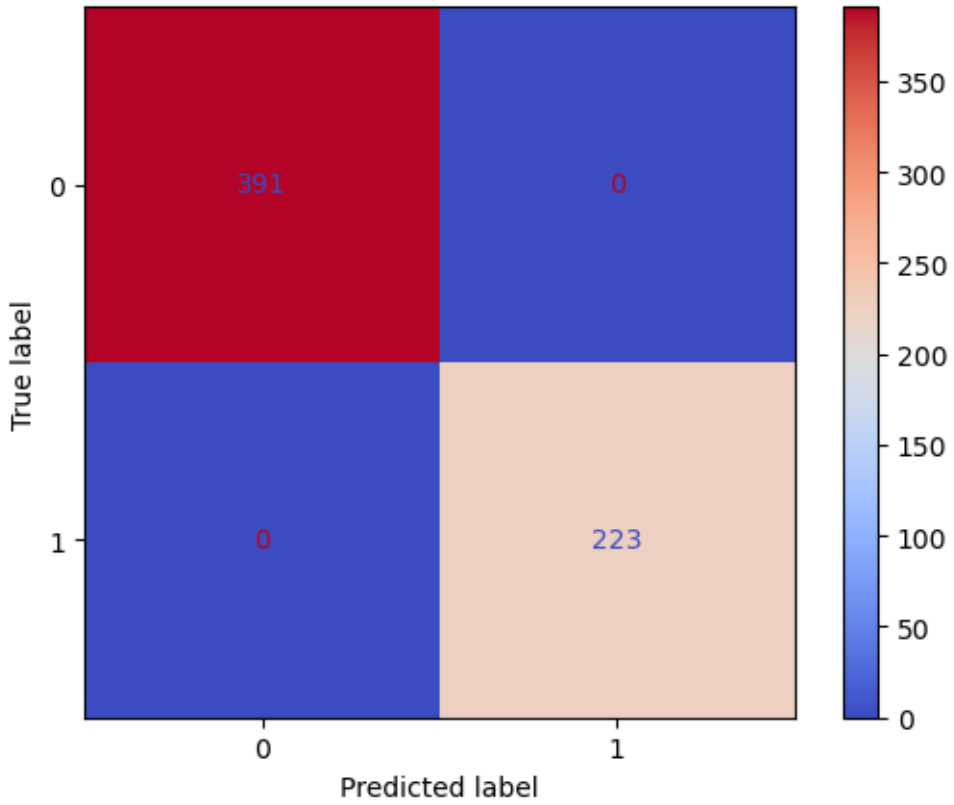
[38]:
```
print(classification_report(y_train,yhat2_train_rfc))
print()
print(classification_report(y_test,yhat2_test_rfc))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       391
           1       1.00      1.00      1.00       223

    accuracy                           1.00       614
   macro avg       1.00      1.00      1.00       614
weighted avg       1.00      1.00      1.00       614


              precision    recall  f1-score   support

           0       0.82      0.82      0.82       109
           1       0.57      0.58      0.57        45

    accuracy                           0.75       154
   macro avg       0.69      0.70      0.70       154
weighted avg       0.75      0.75      0.75       154
```

[39]:
```
print(confusion_matrix(y_train,yhat2_train_rfc))
print()
print(confusion_matrix(y_test,yhat2_test_rfc))
```

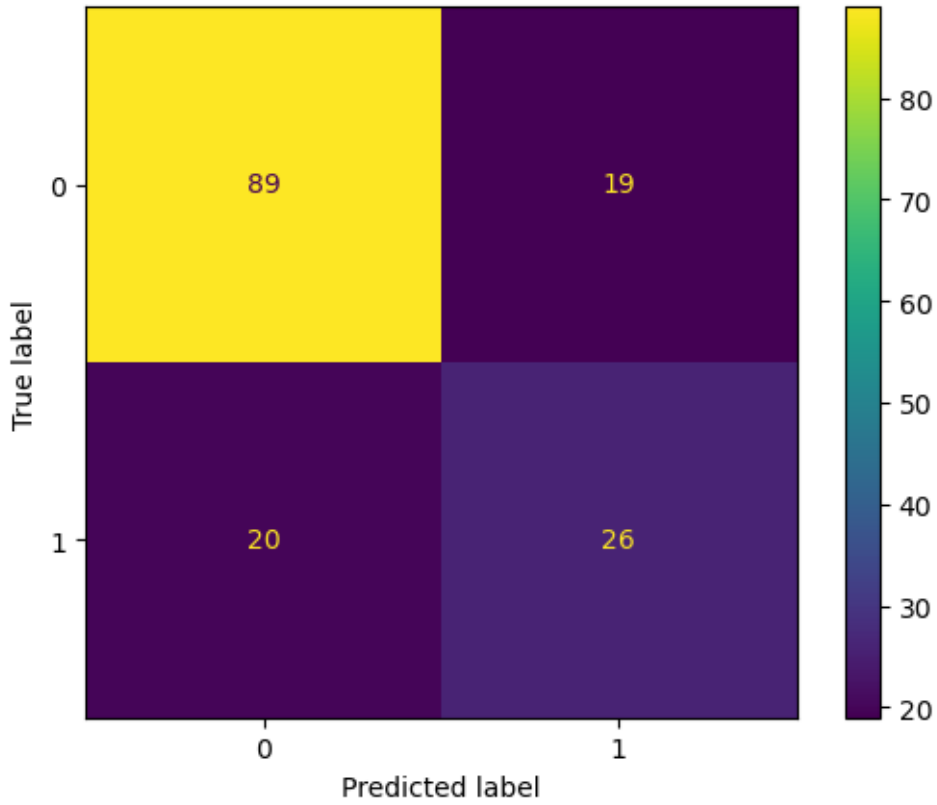
```
[[391   0]
 [  0 223]]

[[89 20]
 [19 26]]
```

[40]:
```
c_matrix=ConfusionMatrixDisplay(confusion_matrix(yhat2_train_rfc,y_train))
c_matrix.plot(cmap='coolwarm')
plt.show()
```



[41]:
```
c_matrix=ConfusionMatrixDisplay(confusion_matrix(yhat2_test_rfc,y_test))
c_matrix.plot(cmap=plt.cm.viridis)
```

[41]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a0c7723b10>



[42]:
```
print('Accuracy: ',accuracy_score(y_train,yhat2_train_rfc))
print()
print('Accuracy: ', accuracy_score(y_test,yhat2_test_rfc))
```

```
Accuracy:  1.0

Accuracy:  0.7467532467532467
```

[43]:
```
scores=cross_val_score(rfc,x,y,cv=3)
np.mean(scores)
```

[43]: 0.7591145833333334

ADA BOOST

[44]:
```
from sklearn.ensemble import AdaBoostClassifier
ada=AdaBoostClassifier()
ada.fit(x_train,y_train)
```

[44]: AdaBoostClassifier()

[45]:
```
yhat3_train_ada=ada.predict(x_train)
yhat3_test_ada=ada.predict(x_test)
```

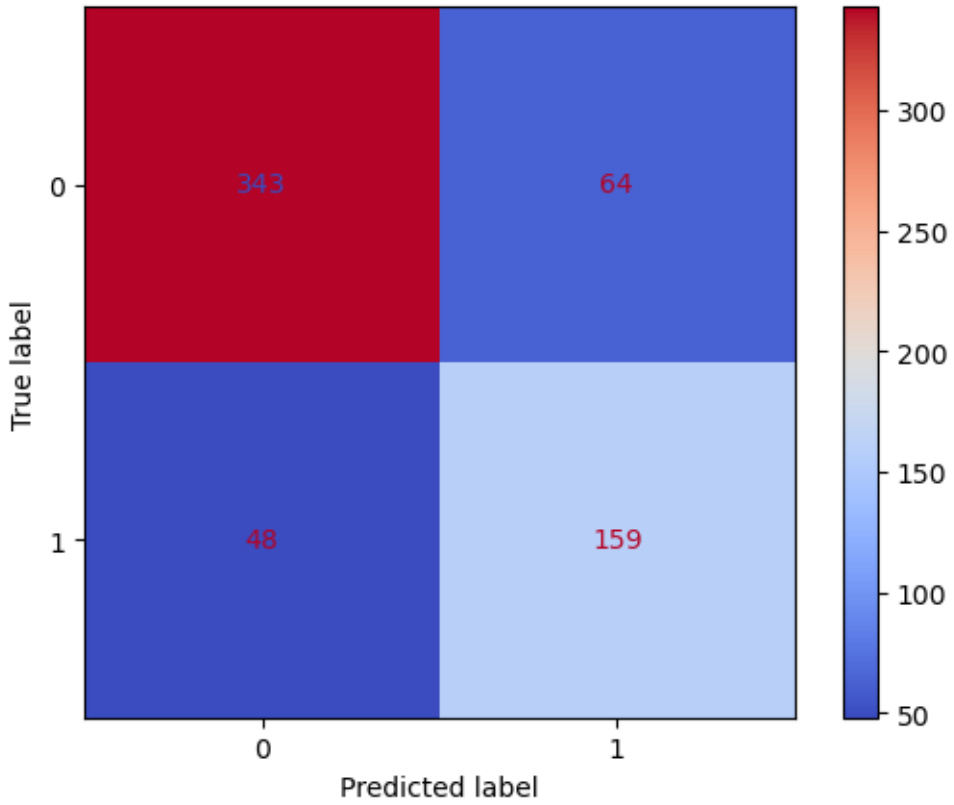
[46]:
```
print(classification_report(y_train,yhat3_train_ada))
print()
print(classification_report(y_test,yhat3_test_ada))
```

```
              precision    recall  f1-score   support

           0       0.84      0.88      0.86       391
           1       0.77      0.71      0.74       223

    accuracy                           0.82       614
   macro avg       0.81      0.80      0.80       614
weighted avg       0.82      0.82      0.82       614


              precision    recall  f1-score   support

           0       0.78      0.83      0.81       109
           1       0.53      0.44      0.48        45

    accuracy                           0.72       154
   macro avg       0.66      0.64      0.65       154
weighted avg       0.71      0.72      0.71       154
```

[47]:
```
print(confusion_matrix(y_train,yhat3_train_ada))
print()
print(confusion_matrix(y_test,yhat3_test_ada))
```

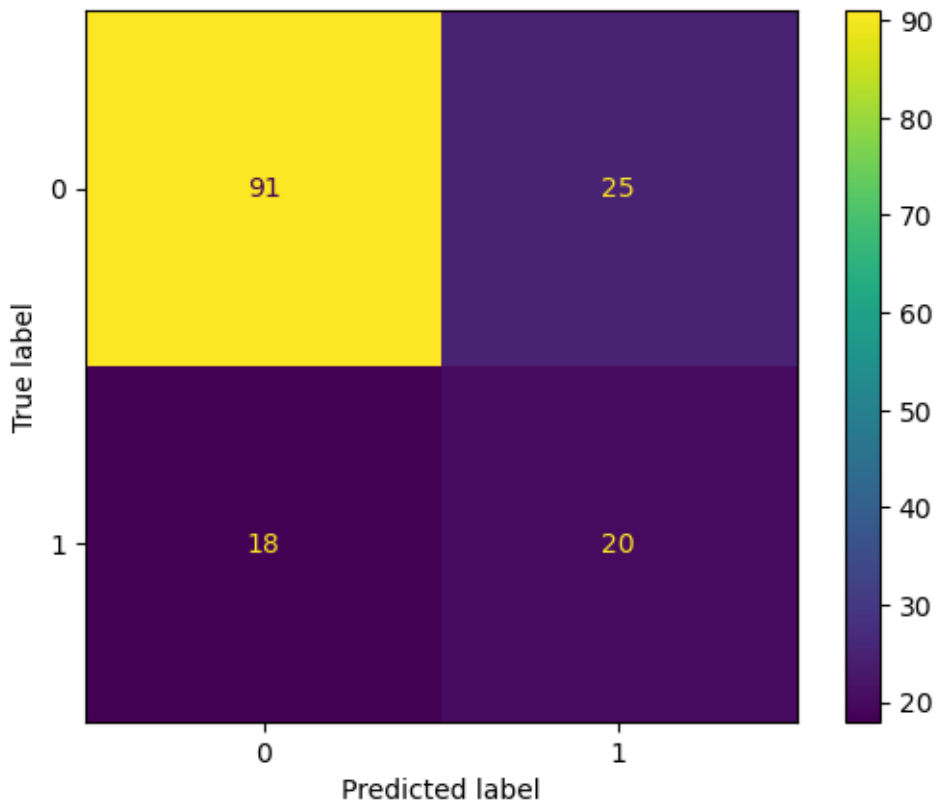
```
[[343  48]
 [ 64 159]]

[[91 18]
 [25 20]]
```

[48]:
```
c_matrix=ConfusionMatrixDisplay(confusion_matrix(yhat3_train_ada,y_train))
c_matrix.plot(cmap='coolwarm')
plt.show()
```



[49]:
```
c_matrix=ConfusionMatrixDisplay(confusion_matrix(yhat3_test_ada,y_test))
c_matrix.plot(cmap=plt.cm.viridis)
```

[49]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a0c5f90310>



[50]:
```
print('Accuracy: ',accuracy_score(y_train,yhat3_train_ada))
print()
print('Accuracy: ', accuracy_score(y_test,yhat3_test_ada))
```

```
Accuracy:  0.8175895765472313

Accuracy:  0.7207792207792207
```

[51]:
```
scores=cross_val_score(ada,x,y,cv=3)
np.mean(scores)
```

[51]:
```
0.7526041666666666
```

XGBOOST

[52]:
```
from xgboost import XGBClassifier
xgb=XGBClassifier()
xgb.fit(x_train,y_train)
```

[52]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
```

```
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, …)
```

[53]:
```
yhat4_train_xgb=xgb.predict(x_train)
yhat4_test_xgb=xgb.predict(x_test)
```

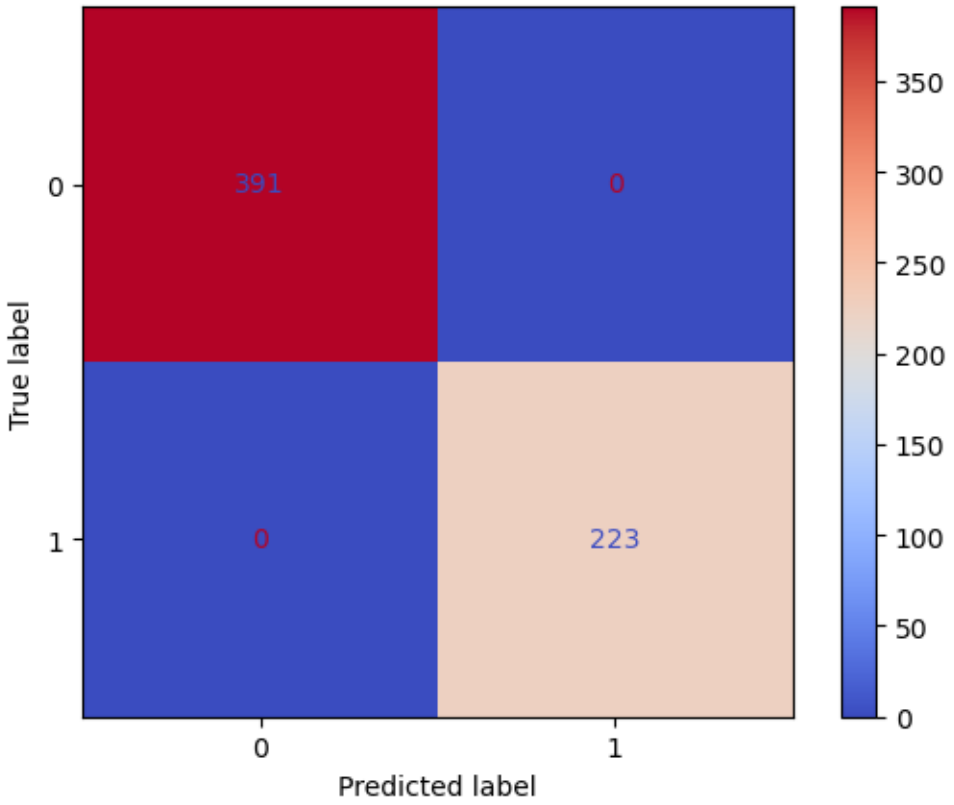
[54]:
```
print(classification_report(y_train,yhat4_train_xgb))
print()
print(classification_report(y_test,yhat4_test_xgb))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       391
           1       1.00      1.00      1.00       223

    accuracy                           1.00       614
   macro avg       1.00      1.00      1.00       614
weighted avg       1.00      1.00      1.00       614


              precision    recall  f1-score   support

           0       0.81      0.78      0.79       109
           1       0.51      0.56      0.53        45

    accuracy                           0.71       154
   macro avg       0.66      0.67      0.66       154
weighted avg       0.72      0.71      0.72       154
```

[55]:
```
print(confusion_matrix(y_train,yhat4_train_xgb))
print()
print(confusion_matrix(y_test,yhat4_test_xgb))
```

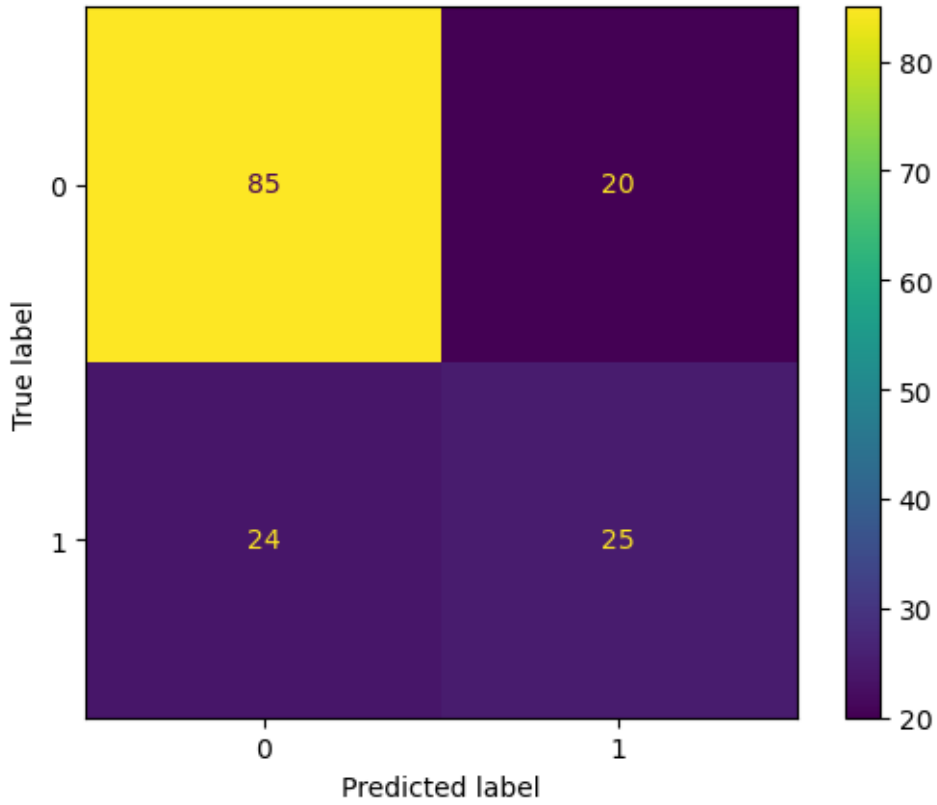
```
[[391   0]
 [  0 223]]

[[85 24]
 [20 25]]
```

[56]:
```
c_matrix=ConfusionMatrixDisplay(confusion_matrix(yhat4_train_xgb,y_train))
c_matrix.plot(cmap='coolwarm')
plt.show()
```



[57]:
```
c_matrix=ConfusionMatrixDisplay(confusion_matrix(yhat4_test_xgb,y_test))
c_matrix.plot(cmap=plt.cm.viridis)
```

[57]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a0c6f745d0>



[58]:
```
print('Accuracy: ',accuracy_score(y_train,yhat4_train_xgb))
print()
print('Accuracy: ', accuracy_score(y_test,yhat4_test_xgb))
```

```
Accuracy:  1.0

Accuracy:  0.7142857142857143
```

[59]:
```
scores=cross_val_score(xgb,x,y,cv=3)
np.mean(scores)
```

[59]: 0.7278645833333334

K NEAREST NEIGHBORS

[60]:
```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier()
knn.fit(x_train,y_train)
```

[60]: KNeighborsClassifier()

[61]:
```
yhat5_train_knn=knn.predict(x_train)
yhat5_test_knn=knn.predict(x_test)
```

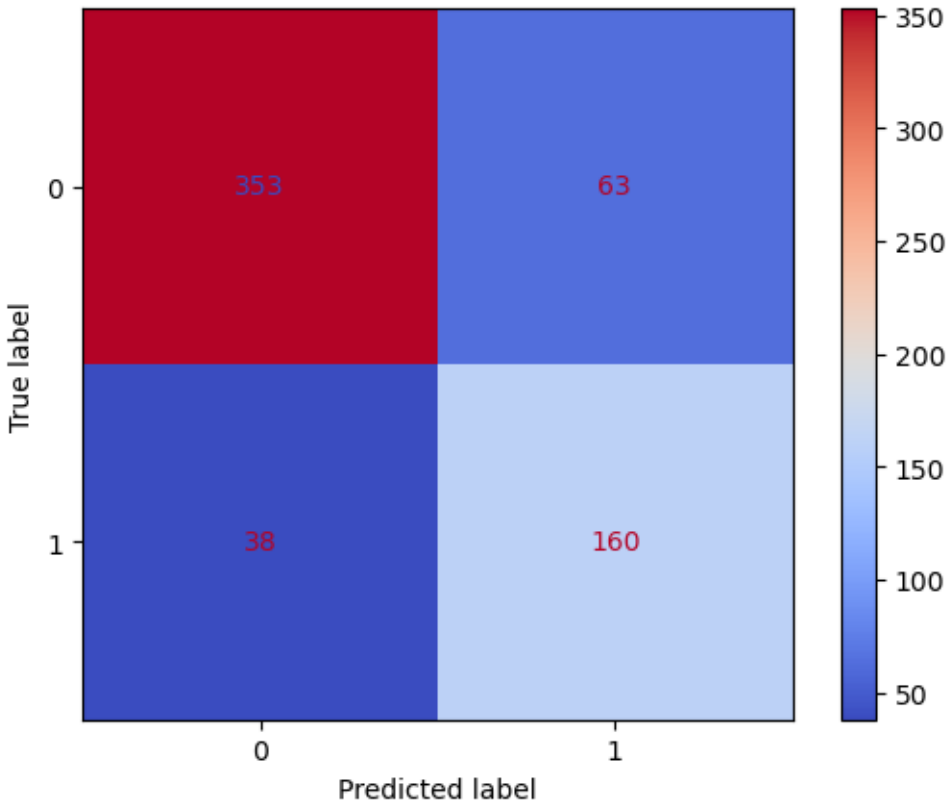
[62]:
```
print(classification_report(y_train,yhat5_train_knn))
print()
print(classification_report(y_test,yhat5_test_knn))
```

```
              precision    recall  f1-score   support

           0       0.85      0.90      0.87       391
           1       0.81      0.72      0.76       223

    accuracy                           0.84       614
   macro avg       0.83      0.81      0.82       614
weighted avg       0.83      0.84      0.83       614


              precision    recall  f1-score   support

           0       0.81      0.84      0.83       109
           1       0.57      0.51      0.54        45

    accuracy                           0.75       154
   macro avg       0.69      0.68      0.68       154
weighted avg       0.74      0.75      0.74       154
```

[63]:
```
print(confusion_matrix(y_train,yhat5_train_knn))
print()
print(confusion_matrix(y_test,yhat5_test_knn))
```

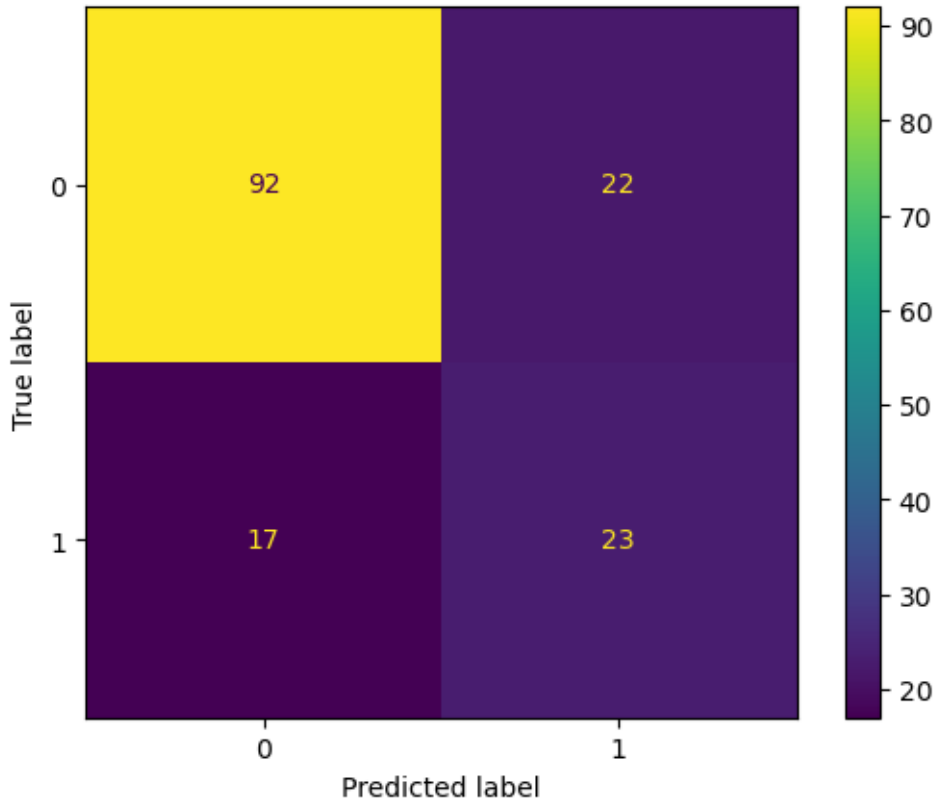
```
[[353  38]
 [ 63 160]]

[[92 17]
 [22 23]]
```

[64]:
```
c_matrix=ConfusionMatrixDisplay(confusion_matrix(yhat5_train_knn,y_train))
c_matrix.plot(cmap='coolwarm')
plt.show()
```



[65]:
```
c_matrix=ConfusionMatrixDisplay(confusion_matrix(yhat5_test_knn,y_test))
c_matrix.plot(cmap=plt.cm.viridis)
```

[65]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a0c7269450>



[66]:
```
print('Accuracy: ',accuracy_score(y_train,yhat5_train_knn))
print()
print('Accuracy: ', accuracy_score(y_test,yhat5_test_knn))
```

```
Accuracy:  0.8355048859934854

Accuracy:  0.7467532467532467
```

[67]:
```
scores=cross_val_score(knn,x,y,cv=3)
np.mean(scores)
```

[67]: 0.7317708333333334

Conclusion

LogisticRegression had the best accuracy of 76% after checking cross validation of the model.