Importing Libraries

In [38]:
```
import pandas as pd
import category_encoders as ce
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score, accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import RocCurveDisplay
from sklearn.preprocessing import MinMaxScaler
from matplotlib import pyplot as plt
import seaborn as sns

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

Reading the Dataset

In [39]:
```
df=pd.read_csv("H:\Heart Disease- Jalpa\heart_2020_cleaned.csv")
Y_COL="HeartDisease"
df.head(5)
```

Out[39]:

	HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	DiffWalking	Sex
0	No	16.60	Yes	No	No	3.0	30.0	No	Female
1	No	20.34	No	No	Yes	0.0	0.0	No	Female
2	No	26.58	Yes	No	No	20.0	30.0	No	Male
3	No	24.21	No	No	No	0.0	0.0	No	Female
4	No	23.71	No	No	No	28.0	0.0	Yes	Female

In [40]:
```
def split_df(df, y_col, random_state, test_size):
    x_cols = [c for c in df.columns if c != y_col]
    X_train, X_test, y_train, y_test = train_test_split(df[x_cols], df[[y_col]],
                                                        stratify=df[[y_col]],
                                                        random_state=random_state, te
    X_train[y_col] = y_train
    X_test[y_col] = y_test
    return X_train.reset_index(drop=True), X_test.reset_index(drop=True)

train_df, test_df = split_df(df, Y_COL, 0, 0.3)
base_ratio = len(train_df[train_df[Y_COL]=='Yes']) / len(train_df)
print(f"{Y_COL} value : {', '.join(train_df[Y_COL].unique())}")
print(f"{Y_COL}=Yes ratio : {base_ratio : .4f}")
```

```
HeartDisease value : No, Yes
HeartDisease=Yes ratio :  0.0856
```

In [41]:
```
df.head()
```

Out[41]:

	HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	DiffWalking	Sex
0	No	16.60	Yes	No	No	3.0	30.0	No	Female
1	No	20.34	No	No	Yes	0.0	0.0	No	Female
2	No	26.58	Yes	No	No	20.0	30.0	No	Male
3	No	24.21	No	No	No	0.0	0.0	No	Female
4	No	23.71	No	No	No	28.0	0.0	Yes	Female

Data Preprocessing: Categorical Variable Encoding

In [42]:
```
t_df = df
binary_cols = ['HeartDisease','Sex','Smoking','AlcoholDrinking','Stroke','Asthma', 'D
for col in binary_cols:
    t_df[col] = t_df[col].replace(list(t_df[col].unique()),[0,1])


race_encoder=ce.OneHotEncoder(cols='Race',handle_unknown='return_nan',return_df=True,
diabetic_encoder = ce.OneHotEncoder(cols='Diabetic', handle_unknown='return_nan', ret
age_encoder= ce.OrdinalEncoder(cols=['AgeCategory'],return_df=True,
                           mapping=[{'col':'AgeCategory',
'mapping':{'18-24':0, '25-29':1,'30-34':2,'35-39':3,'40-44':4,'45-49':5,'50-54':6,'55
health_encoder = ce.OrdinalEncoder(cols=['GenHealth'], return_df=True,
                                    mapping=[{'col':'GenHealth',
                                            'mapping':{'Poor':0,'Fair':1,'Good':2,'Ve

t_df = age_encoder.fit_transform(t_df)
t_df = health_encoder.fit_transform(t_df)
t_df = race_encoder.fit_transform(t_df)
t_df = diabetic_encoder.fit_transform(t_df)
t_df.head(5)
```

Out[42]:

	HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	DiffWalking	Sex	Age
0	0	16.60	0	0	0	3.0	30.0	0	0	
1	0	20.34	1	0	1	0.0	0.0	0	0	
2	0	26.58	0	0	0	20.0	30.0	0	1	
3	0	24.21	1	0	0	0.0	0.0	0	0	
4	0	23.71	1	0	0	28.0	0.0	1	0	

5 rows × 26 columns

Data Preprocessing: Categorical Variable Encoding

In [43]:
```
scaler = MinMaxScaler()
names = t_df.columns
d = scaler.fit_transform(t_df)
scaled_df = pd.DataFrame(d, columns=names)
scaled_df.head()
```

Out[43]:

	HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	DiffWalking	Sex
0	0.0	0.055294	0.0	0.0	0.0	0.100000	1.0	0.0	0.0
1	0.0	0.100447	1.0	0.0	1.0	0.000000	0.0	0.0	0.0
2	0.0	0.175782	0.0	0.0	0.0	0.666667	1.0	0.0	1.0
3	0.0	0.147169	1.0	0.0	0.0	0.000000	0.0	0.0	0.0
4	0.0	0.141132	1.0	0.0	0.0	0.933333	0.0	1.0	0.0

5 rows × 26 columns

Dataset Balancing through Oversampling

In [44]:
```
class_0 = scaled_df[scaled_df['HeartDisease'] == 0]
class_1 = scaled_df[scaled_df['HeartDisease'] == 1]
class_1 = class_1.sample(len(class_0),replace=True)
train_df = pd.concat([class_0, class_1], axis=0)
print('Data in Train:')
print(train_df['HeartDisease'].value_counts())
```

```
Data in Train:
HeartDisease
0.0    292422
1.0    292422
Name: count, dtype: int64
```

Model Evaluation Metrics

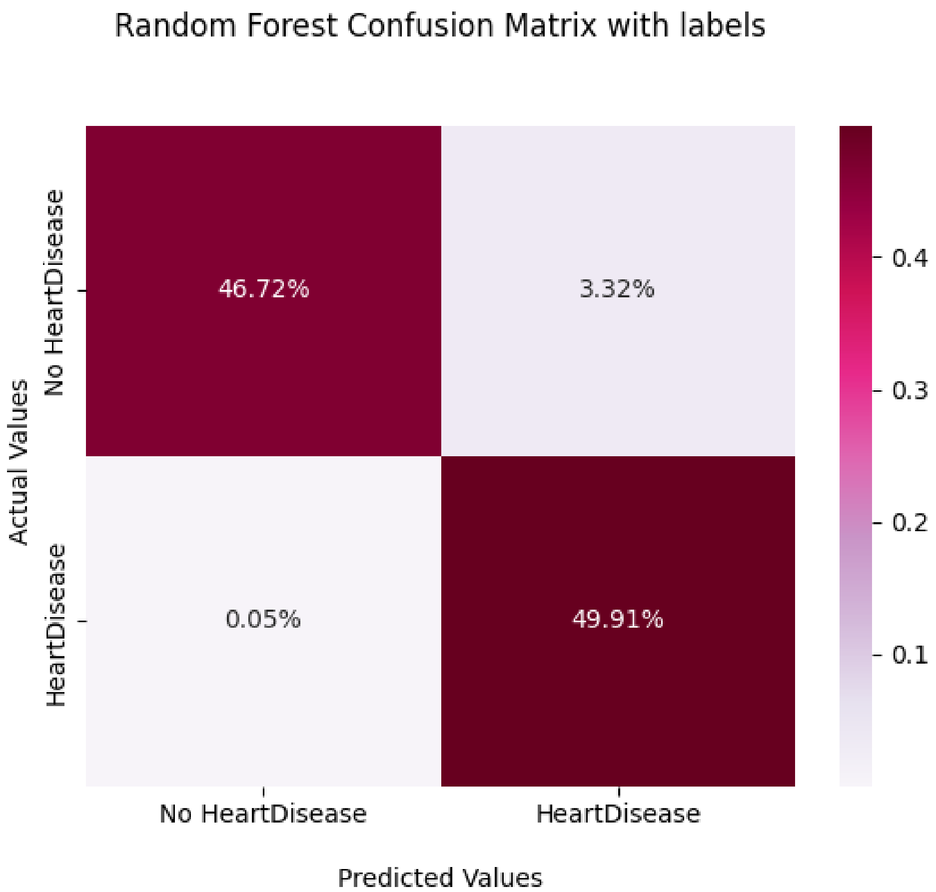
In [45]:
```
x = train_df[['AgeCategory','DiffWalking','Stroke','Diabetic_Yes','Diabetic_No','Kidn
y = train_df['HeartDisease']
x_train, x_test, y_train, y_test =train_test_split(x,y,test_size=0.2, random_state=42
```

Random Forest Classifier

In [46]:
```
clf = RandomForestClassifier(n_estimators = 100)
clf.fit(x_train, y_train)
clf_y_predict = clf.predict(x_test)
print(f'model: {str(clf)}')
print(f'Accuracy_score: {accuracy_score(y_test,clf_y_predict)}')
print(f'Precission_score: {precision_score(y_test,clf_y_predict)}')
print(f'Recall_score: {recall_score(y_test,clf_y_predict)}')
print(f'F1-score: {f1_score(y_test,clf_y_predict)}')
```

```
model: RandomForestClassifier()
Accuracy_score: 0.9663244107413075
Precission_score: 0.9376676356744776
Recall_score: 0.9990075290896646
F1-score: 0.967366179796691
```

In [47]:
```
cm  = confusion_matrix(y_test, clf_y_predict)
ax = sns.heatmap(cm/np.sum(cm), annot=True, cmap='PuRd', fmt='.2%')
ax.set_title('Random Forest Confusion Matrix with labels\n\n');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');
ax.xaxis.set_ticklabels(['No HeartDisease','HeartDisease'])
ax.yaxis.set_ticklabels(['No HeartDisease','HeartDisease'])
plt.show()
```

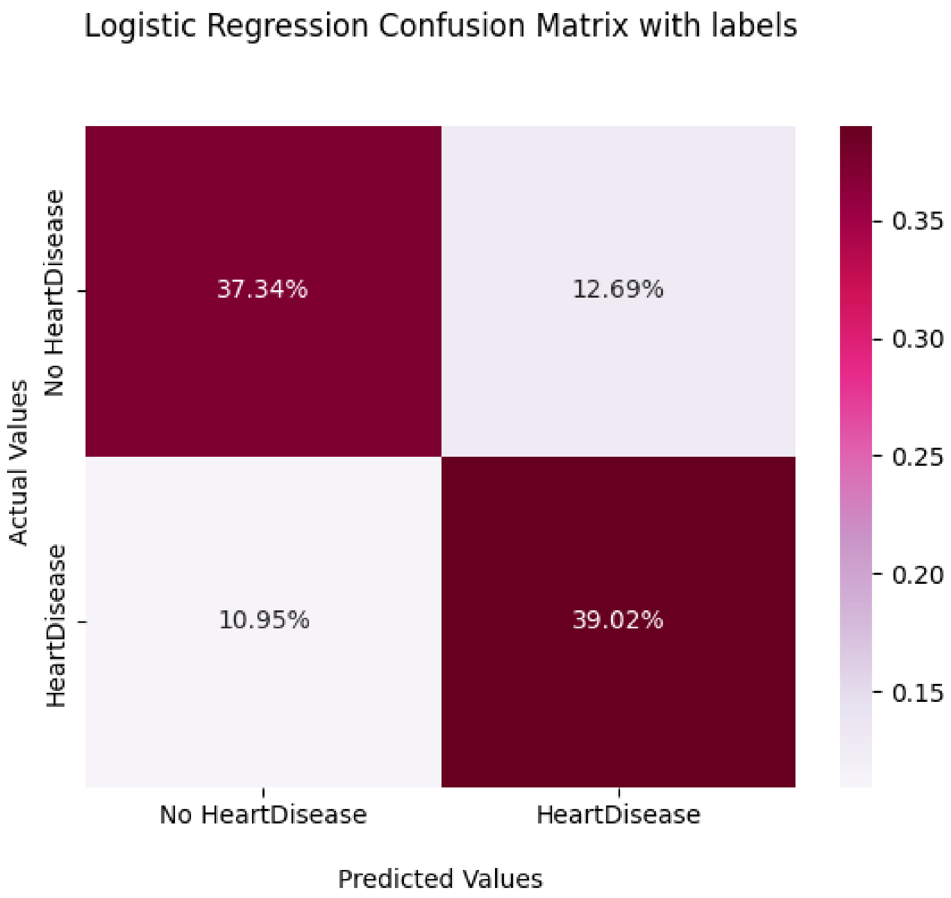


Logistic Regression

In [48]:
```
lr = LogisticRegression(random_state=0)
lr.fit(x_train, y_train)
lr_y_predict = lr.predict(x_test)
print(f'model: {str(lr)}')
print(f'Accuracy_score: {accuracy_score(y_test,lr_y_predict)}')
print(f'Precission_score: {precision_score(y_test,lr_y_predict)}')
print(f'Recall_score: {recall_score(y_test,lr_y_predict)}')
print(f'F1-score: {f1_score(y_test,lr_y_predict)}')
```

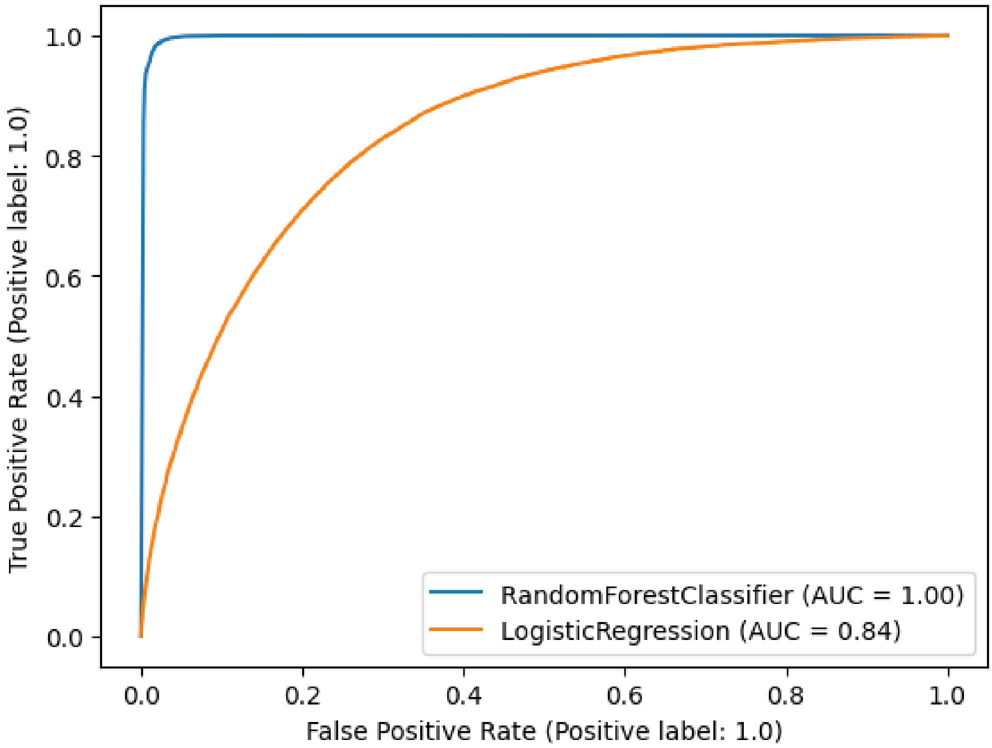
```
model: LogisticRegression(random_state=0)
Accuracy_score: 0.7636125811112346
Precission_score: 0.7545301236690695
Recall_score: 0.7809206023271732
F1-score: 0.7674985705156234
```

In [49]:
```
cm  = confusion_matrix(y_test, lr_y_predict)
ax = sns.heatmap(cm/np.sum(cm), annot=True, cmap='PuRd', fmt='.2%')
ax.set_title('Logistic Regression Confusion Matrix with labels\n\n');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');
ax.xaxis.set_ticklabels(['No HeartDisease','HeartDisease'])
ax.yaxis.set_ticklabels(['No HeartDisease','HeartDisease'])
plt.show()
```



ROC Curve

In [50]:
```
ax = plt.gca()
rdf_disp = RocCurveDisplay.from_estimator(clf, x_test, y_test, ax=ax)
lg_disp = RocCurveDisplay.from_estimator(lr, x_test, y_test, ax=ax)
plt.show()
```



In [ ]: