Author:kkitonga
Date: 11/11/2022
Classification :k-nearest neigbors and logistic regression

import libraries and necessary modules

In [1]:
```
from sklearn.neighbors import KNeighborsClassifier                              #fitting k-nearest neighbo
from sklearn.model_selection import train_test_split                            #splitting data:training a
from sklearn.metrics import precision_recall_curve                              #plotting tradeoff :recall
import pandas as pd                                                             #managing dataframes
import numpy as np                                                              #manipulation
from sklearn.linear_model import LogisticRegression                             #fitting logistic regressi
from sklearn import metrics
from sklearn.metrics import confusion_matrix,classification_report              #metrics to assess classif
import matplotlib.pyplot as plt                                                 #visualization
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

1.load data

In [2]:
```
data = pd.read_csv("C:/Users/Karengi/Desktop/MIT/diabetes.csv")
```

2.read data

In [3]:
```
data.head()
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	79	33.6	0.627	50	1
1	1	85	66	29	79	26.6	0.351	31	0
2	8	183	64	20	79	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

In [4]:
```
data.tail()
```

Out[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	79	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	20	79	30.1	0.349	47	1
767	1	93	70	31	79	30.4	0.315	23	0

In [5]:
```
data.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
```

In [6]:
```
data.shape
```

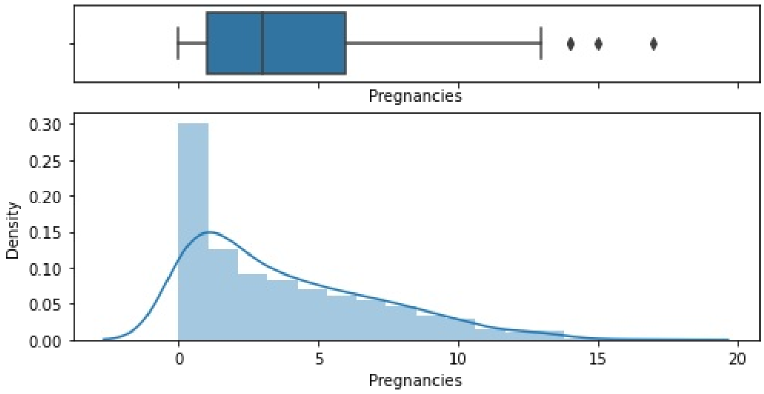
Out[6]:
```
(768, 9)
```

3.Visualization

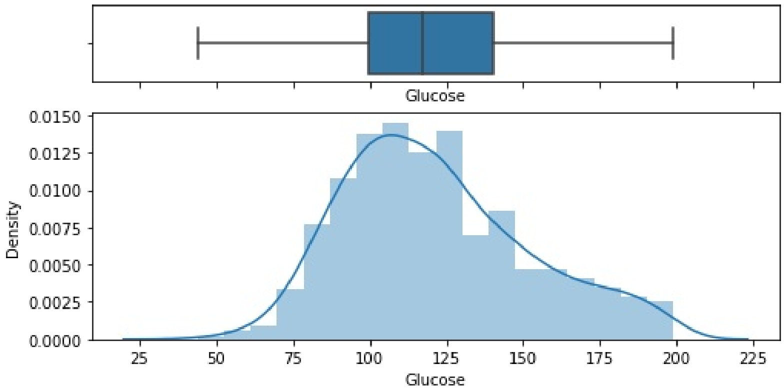
In [7]:
```
for i in data.iloc[:,[0,1,2,3,4,5,6,7]]:
    f,(ax_box,ax_hist) =plt.subplots(2,sharex=True,gridspec_kw={'height_ratios':(0.25,0.75)},figsize=(8,4))
    print(i)
    sns.boxplot(data[i],ax=ax_box)
```

```
    sns.distplot(data[i],ax=ax_hist)
    plt.show()
```

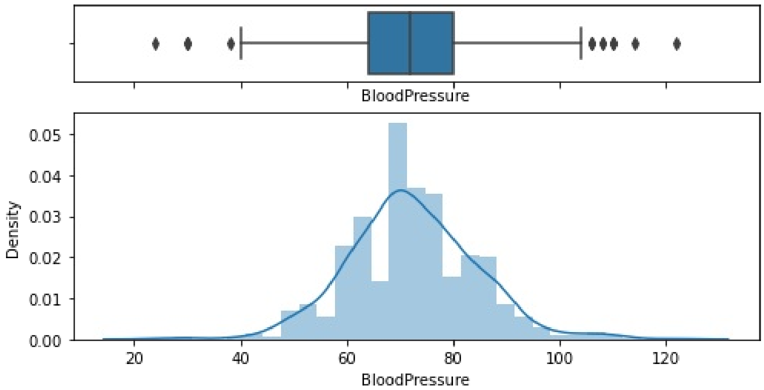
```
Pregnancies
```



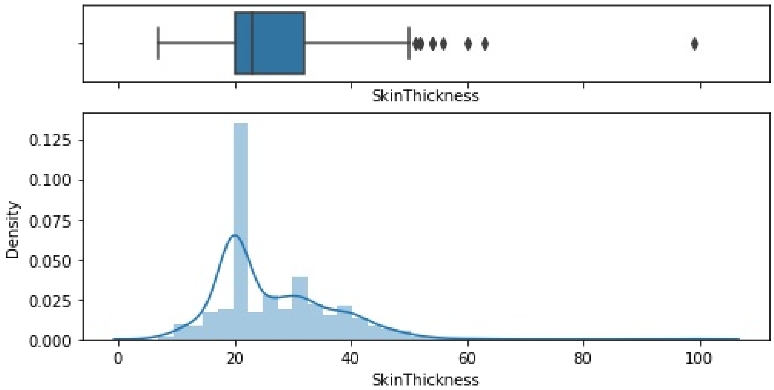

```
Glucose
```



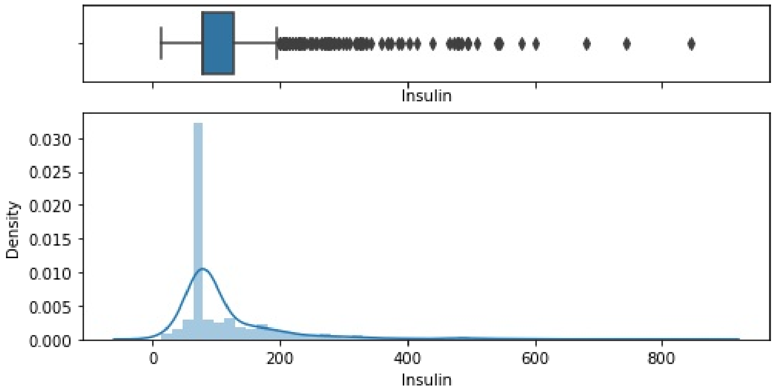

```
BloodPressure
```



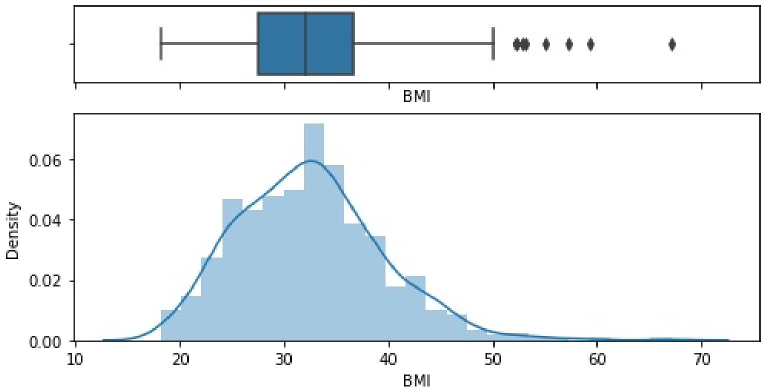

```
SkinThickness
```



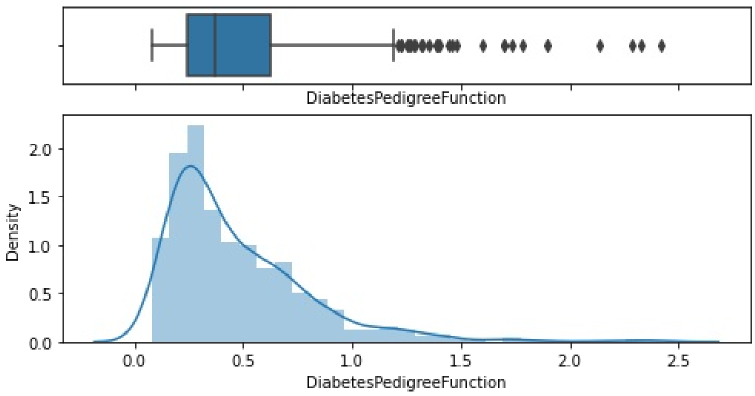

```
Insulin
```



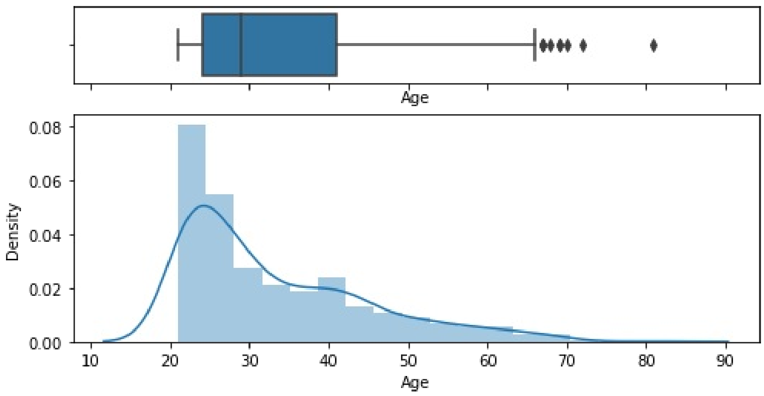
BMI



DiabetesPedigreeFunction



Age



4.Preparing training and test dataset

In [8]:
```
X = data.drop('Outcome',axis=1).values                                  #object with only features(inde
```

In [9]:
```
y = data.Outcome.values                                                 #object with target variable onl
```

```
In [9]: y = data.Outcome.values                                          #object with target variable ont
```

```
In [10]: X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=21,stratify=y) #dataset splitti
```

5.Classification :k-nearest neighbors

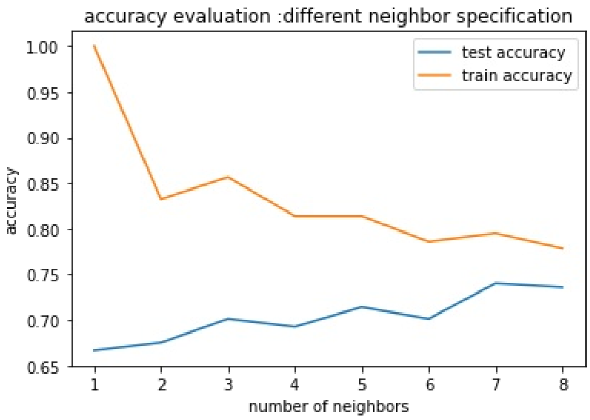
i.K-nearest neighbors with specification of n_neighbors ranging from 1 to 9

```
In [11]: jirani=np.arange(1,9)                                                    #specifiying different values for
         train_accuracy = np.empty(len(jirani))       #object to hold values for accuracy on trainset for all n_neigbor
         test_accuracy = np.empty(len(jirani))        #object to hold values for accuracy on testset for all n_neigbor s

         for i,j in enumerate(jirani) :
             knn = KNeighborsClassifier(j)                                      #creating object for fitting KNeighborsC
             knn.fit(X_train,y_train)                                           #fiting k-nearest neighbors
             train_accuracy[i] =knn.score(X_train,y_train)                      #accuracy score on train data
             test_accuracy[i] =knn.score(X_test,y_test)                         #accuracy on test data
```

ii.evaluation of accuracy across different n_neigbors specifications

```
In [12]: plt.title('accuracy evaluation :different neighbor specification')
         plt.plot(jirani,test_accuracy,label='test accuracy')
         plt.plot(jirani,train_accuracy,label='train accuracy')
         plt.legend()
         plt.xlabel('number of neighbors')
         plt.ylabel('accuracy')
         plt.show()
```



6.Classification: logistic regression

i.fitting logistic regression

```
In [13]: lg = LogisticRegression()                                              #creating object for logisitc regression
         lg.fit(X_train, y_train)                                               #fitting logistic regression
```

```
Out[13]: LogisticRegression()
```

ii.prediction on train set

```
In [14]: y_pred_train = lg.predict(X_train)
         print(classification_report(y_train, y_pred_train))
```

```
              precision    recall  f1-score   support

           0       0.82      0.89      0.85       350
           1       0.75      0.64      0.69       187

    accuracy                           0.80       537
   macro avg       0.79      0.76      0.77       537
weighted avg       0.80      0.80      0.80       537
```

iii.prediction on test set

```
In [15]: y_pred_test = lg.predict(X_test)
         print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.76      0.88      0.81       150
           1       0.68      0.48      0.57        81

    accuracy                           0.74       231
   macro avg       0.72      0.68      0.69       231
weighted avg       0.73      0.74      0.73       231
```

In [16]:
```
print(metrics.accuracy_score(y_test, y_pred_test))
```

```
0.7402597402597403
```

iv.predicting probabilities on train set

In [17]:
```
y_scores_lg=lg.predict_proba(X_train)
```

In [18]:
```
print(y_scores_lg)                                          #see output from y_scores_lg
```

```
[[0.86900413 0.13099587]
 [0.25331722 0.74668278]
 [0.42488165 0.57511835]
 ...
 [0.86929797 0.13070203]
 [0.31331892 0.68668108]
 [0.66528652 0.33471348]]
```

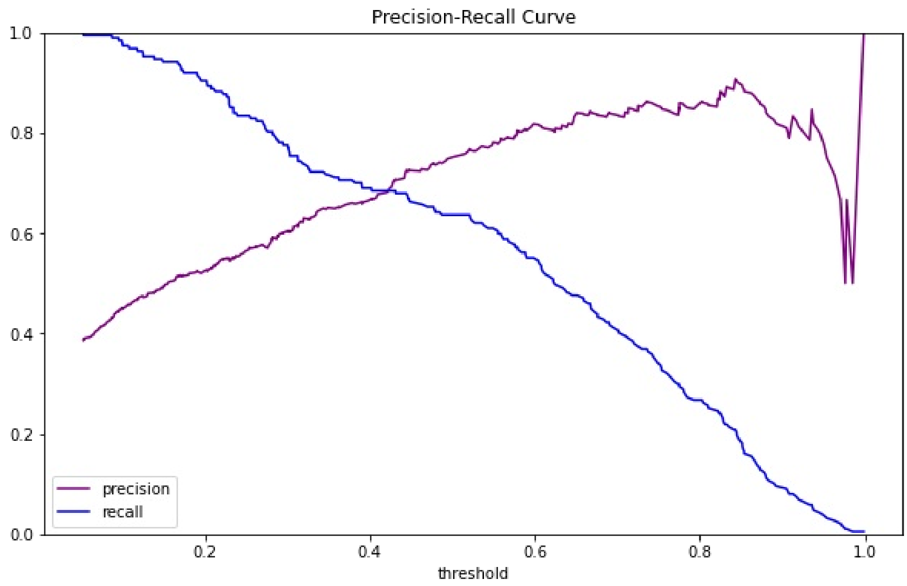
v.precision recall curve

In [19]:
```
#i.precision,recall,thresholds values
precision, recall, thresholds = precision_recall_curve(y_train, y_scores_lg[:,1]) #[:,1]:references second colu

#Precision recall curve
#i.Plot specification
fig, ax = plt.subplots(figsize=(10,6))
ax.plot(thresholds,precision[:-1], color='purple',label='precision')
ax.plot(thresholds,recall[:-1],color='blue',label='recall')

#ii.adding axis labels to plot
ax.set_title('Precision-Recall Curve')
ax.set_xlabel('threshold')

#iii.display plot
plt.ylim([0,1])
plt.legend(loc='lower left')
plt.show()
```



vi.adjusting threshold

In [ ]:
```
threshold = 0.35                            #selected 0.35 based on trying to increase recall but not lose out
```

vii.assesing classification after adjusting threshold on training data

In [21]:
```
y_pred_train = lg.predict_proba(X_train)
metrics.accuracy_score(y_train, y_pred_train[:,1]>threshold)
```

Out[21]:
```
0.7653631284916201
```

viii.assesing classification after adjusting threshold on test data¶

Assessing classification after adjusting threshold on test data

In [22]:
```
y_pred_test = lg.predict_proba(X_test)
metrics.accuracy_score(y_test, y_pred_test[:,1]>threshold)
```

Out[22]:
```
0.7532467532467533
```

End

In [ ]:
```
```