MT2022161 ADITYA.M

ASSIGNMENT -1 [AI - 511]

PREPROCESSING

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
d=pd.read_csv("breast_cancer.csv")
```

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

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	p
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	

5 rows × 33 columns

In [4]:
```
d.isna().sum()
```

Out[4]:
```
id                           0
diagnosis                    0
radius_mean                  0
texture_mean                 0
perimeter_mean               0
area_mean                    0
smoothness_mean              0
compactness_mean             0
concavity_mean               0
concave points_mean          0
symmetry_mean                0
fractal_dimension_mean       0
radius_se                    0
texture_se                   0
perimeter_se                 0
area_se                      0
smoothness_se                0
compactness_se               0
concavity_se                 0
concave points_se            0
symmetry_se                  0
fractal_dimension_se         0
radius_worst                 0
texture_worst                0
perimeter_worst              0
area_worst                   0
smoothness_worst             0
compactness_worst            0
concavity_worst              0
concave points_worst         0
symmetry_worst               0
fractal_dimension_worst      0
Unnamed: 32                569
dtype: int64
```

Last column and the 'id' column are not useful , hence we can get rid of it.

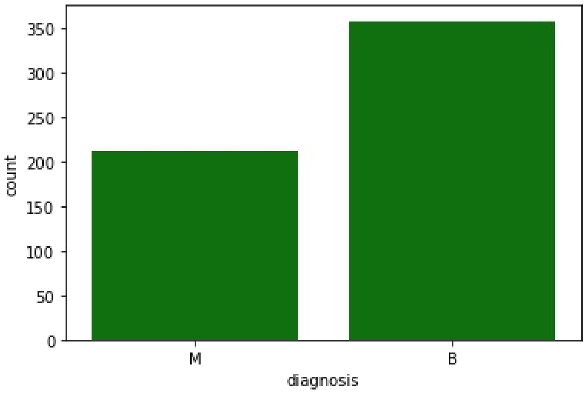
In [5]:
```
d.drop(d.columns[[0,-1]], axis=1, inplace=True)
```

In [6]:
```
d['diagnosis'].value_counts()
```

Out[6]:
```
B    357
M    212
Name: diagnosis, dtype: int64
```

In [7]:
```
import seaborn as sns
sns.countplot(x='diagnosis', data=d, color='green')
```

Out[7]:
```
<AxesSubplot:xlabel='diagnosis', ylabel='count'>
```



There are sufficinet number of data points in both the classes to train the model properly.

In [8]:
```
d.dtypes
```

Out[8]:
```
diagnosis                   object
radius_mean                float64
texture_mean               float64
perimeter_mean             float64
area_mean                  float64
smoothness_mean            float64
compactness_mean           float64
concavity_mean             float64
concave points_mean        float64
symmetry_mean              float64
fractal_dimension_mean     float64
radius_se                  float64
texture_se                 float64
perimeter_se               float64
area_se                    float64
smoothness_se              float64
compactness_se             float64
concavity_se               float64
concave points_se          float64
symmetry_se                float64
fractal_dimension_se       float64
radius_worst               float64
texture_worst              float64
perimeter_worst            float64
area_worst                 float64
smoothness_worst           float64
compactness_worst          float64
concavity_worst            float64
concave points_worst       float64
symmetry_worst             float64
fractal_dimension_worst    float64
dtype: object
```

label encoding

In [9]:
```
#encoding the 'catogery' column into a numeric format
for i,classification in enumerate(d.iloc[:,0].values):
    if classification =='M':
        d.iloc[i,0]=1
    else:
        d.iloc[i,0]=0
```

standerdization of data

In [10]:
```
#getting the input features and output variable from the data set after standerdization
y=d.iloc[:,0]
for feature in d.columns[1:]:
    mean=d[feature].mean()
    std=d[feature].std()
    min=d[feature].min()
    max=d[feature].max()
    d[feature]=((d[feature]-mean)/(std))
x=d.iloc[:,1:] #standerdized values
x=x.values
y=y.values
```

data splitting

In [11]:
```
# splitting the data into train test
import random
def split(X,Y,ratio=0.4):
    n=int(ratio*len(Y))
```

```
    #train size is 40% by default
    train_index=[]
    test_index=[]

    while len(train_index)!=n:
        k=random.randrange(0,len(X))
        if k not in train_index:
            train_index.append(k)
        else:
            pass
    train_index.sort()

    for i in range(0,len(X)):
        if i not in train_index:
            test_index.append(i)

    x_train=[]
    y_train=[]
    for index in train_index:
        x_train.append(X[index])
        y_train.append(Y[index])

    x_test=[]
    y_test=[]
    for index in test_index:
        x_test.append(X[index])
        y_test.append(Y[index])
    return x_test,x_train,y_test,y_train
```

In [12]:
```
x_test,x_train,y_test,y_train=split(x,y,0.7) #70% of dataset for training
x_test=np.array(x_test)
x_train=np.array(x_train)
y_test=np.array(y_test)
y_train=np.array(y_train)
```

Taking x and y in np arrays for matrix multiplication

In [13]:
```
x_test=x_test.T
x_train=x_train.T
y_test=y_test.reshape(1,x_test.shape[1])
y_train=y_train.reshape(1,x_train.shape[1])
print(x_test.shape)
print(y_test.shape)
print(x_train.shape)
print(y_train.shape)
```

```
(30, 171)
(1, 171)
(30, 398)
(1, 398)
```

LOGISTIC REGRESSION

STEP 1: defining the model class

In [14]:
```
class logisticRegression():
    def __init__(self):
        self.w=None # w1 till wk
        self.b=None # w0 ----> y=w0+w1x1+w2x2+.......

    def sigmoid(self,t):
        return 1/(1+np.exp(-t))

    def fit(self,X,Y,lr,iter):
        self.w=None
        self.b=None
        n,k=X.shape[1],X.shape[0] #n data points given and k features
        self.w=np.zeros((k,1)) #weights for all the features
        self.b=0 #constant parameter
        while(iter):
            iter-=1
            line=np.dot(self.w.T,X)+self.b #wT*x is being calculated
            yp=self.sigmoid(line) # sigmoid(wT*x)
            dW=(1/n)*np.dot(Y-yp,X.T)
            dB=(1/n)*np.sum(Y-yp)
            self.w=self.w+lr*dW.T #moving in the direction of gradient to maximize cost
            self.b=self.b+lr*dB

    def predict(self,x):
        y=np.dot(self.w.T,x)+self.b
        y_pred=self.sigmoid(y)
        n=len(y_pred[0])
        for i in range(n):
            if y_pred[0][i]>0.5:
```

```
                y_pred[0][i]=1
            else:
                y_pred[0][i]=0
        return y_pred

    def validate(self,y_pred,y_test):
        true_p=true_n=false_n=false_p=0
        n=len(y_pred[0])
        for i in range(n):
            if y_pred[0][i]==y_test[0][i]:
                if y_pred[0][i]==1:
                    true_p+=1
                else:
                    true_n+=1
            else:
                if y_pred[0][i]==1:
                    false_p+=1
                else:
                    false_n+=1
        #precision is nothing but the total number of correct positive predictions /all predicted positive
        precision=(true_p)/(true_p+false_p)
        #recall is equal to number of correct positive predictions / number of actual positives
        recall=(true_p)/(true_p+false_n)
        #harmonic mean of recall and precision
        f1score=(2*precision*recall)/(recall+precision)
        print("----------------Confusion Matrix----------------")
        print("total number of observations:",n)
        print("true positive:",true_p)
        print("true negative:",true_n)
        print("false positive:",false_p)
        print("false negative:",false_n)
        print("------------------------------------------------")
        print("precision of your model is:",precision)
        print("recall of your model is:",recall)
        print("f1score of your model is:",f1score)
        print("accuracy of your model is:",(true_p+true_n)/n)
```

Step 2: Gradient Ascent predection fot testing data

In [15]:
```
mod2=logisticRegression()
mod2.fit(x_train,y_train,0.001,10000)
y_pred=mod2.predict(x_test)
mod2.validate(y_pred,y_test)
```

```
----------------Confusion Matrix----------------
total number of observations: 171
true positive: 66
true negative: 101
false positive: 1
false negative: 3
------------------------------------------------
precision of your model is: 0.9850746268656716
recall of your model is: 0.9565217391304348
f1score of your model is: 0.9705882352941176
accuracy of your model is: 0.9766081871345029
```

In [16]:
```
from sklearn.metrics import confusion_matrix, f1_score
print(confusion_matrix(y_test.T,y_pred.T))
print(f1_score(y_test.T,y_pred.T))
```

```
[[101   1]
 [  3  66]]
0.9705882352941176
```

Naive Bayes Classifier

Naive Bayes Classifier is based on the Naive assumption which assums that all the features in the dataset and independent of each other and are equally important.

Data

In [17]:
```
d.head(10)
```

Out[17]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean
0	1	1.096100	-2.071512	1.268817	0.983510	1.567087	3.280628	2.650542	2.530249
1	1	1.828212	-0.353322	1.684473	1.907030	-0.826235	-0.486643	-0.023825	0.547662
2	1	1.578499	0.455786	1.565126	1.557513	0.941382	1.052000	1.362280	2.035440
3	1	-0.768233	0.253509	-0.592166	-0.763792	3.280667	3.399917	1.914213	1.450431
4	1	1.748758	-1.150804	1.775011	1.824624	0.280125	0.538866	1.369806	1.427237
5	1	-0.475956	-0.834601	-0.386808	-0.505206	2.235455	1.243242	0.865540	0.823931
6	1	1.169878	0.160508	1.137124	1.094332	-0.123028	0.088218	0.299809	0.646366
7	1	-0.118413	0.358135	-0.072803	-0.218772	1.602639	1.139100	0.060972	0.281702
8	1	-0.319885	0.588312	-0.183919	-0.383870	2.199903	1.682529	1.218025	1.149680
9	1	-0.473118	1.104467	-0.329192	-0.508616	1.581308	2.561105	1.737343	0.940932

10 rows × 31 columns

In [18]:

```
d.describe()
```

Out[18]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean
count	5.690000e+02	5.690000e+02	5.690000e+02	5.690000e+02	5.690000e+02	5.690000e+02	5.690000e+02	5.690000e+02
mean	-3.142575e-15	-6.558316e-15	-7.012551e-16	-8.339355e-16	6.083788e-15	-1.081346e-15	-3.703345e-16	9.935423e-16
std	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00
min	-2.027864e+00	-2.227289e+00	-1.982759e+00	-1.453164e+00	-3.109349e+00	-1.608721e+00	-1.113893e+00	-1.260710e+00
25%	-6.887793e-01	-7.253249e-01	-6.913472e-01	-6.666089e-01	-7.103378e-01	-7.464292e-01	-7.430941e-01	-7.372951e-01
50%	-2.148925e-01	-1.045442e-01	-2.357726e-01	-2.949274e-01	-3.486040e-02	-2.217454e-01	-3.419391e-01	-3.973715e-01
75%	4.689800e-01	5.836621e-01	4.992377e-01	3.631877e-01	6.356397e-01	4.934227e-01	5.255994e-01	6.463664e-01
max	3.967796e+00	4.647799e+00	3.972634e+00	5.245913e+00	4.766717e+00	4.564409e+00	4.239858e+00	3.924477e+00

8 rows × 30 columns

In [19]:

```
x_train.shape # 30 features and 341 data points......... 30 rows
```

Out[19]:

```
(30, 398)
```

Defining the model class for binary NaiveBayes

In [20]:

```
class binaryNaiveBayes():
    def __init__(self):
        self.mean0=None #mean for class 0
        self.std0=None #standerd deviation for class  0
        self.mean1=None
        self.std1=None
        self.countzero=None # number of data points of class 0 in training data
        self.countone=None
        self.n=None

    def fit(self,x,y):
        x=x.T
        y=y.T
        self.n=len(y) #number of data points
        self.countone=0
        self.countzero=0
        k=x.shape[1] #number of features
        self.mean0=[0 for i in range(k)]
        self.mean1=[0 for i in range(k)]
        self.std0=[0 for i in range(k)]
        self.std1=[0 for i in range(k)]
        for i in range(self.n):  #calculating mean for all the features
            if y[i][0]==1:
                self.countone=self.countone+1
                for j in range(k):
                    self.mean1[j]=self.mean1[j]+x[i][j]
            else:
                self.countzero=self.countzero+1
                for j in range(k):
                    self.mean0[j]=self.mean0[j]+x[i][j]

        for j in range(k):
            self.mean1[j]=self.mean1[j]/self.countone
            self.mean0[j]=self.mean0[j]/self.countzero
```

```python
        for i in range(self.n):  #calculating standerd deviation for all the features
            if y[i][0]==1:
                for j in range(k):
                    self.std1[j]=self.std1[j]+((x[i][j]-self.mean1[j])*(x[i][j]-self.mean1[j]))

            else:
                for j in range(k):
                    self.std0[j]=self.std0[j]+((x[i][j]-self.mean0[j])*(x[i][j]-self.mean0[j]))

        for j in range(k):
            self.std0[j]=self.std0[j]/self.countzero
            self.std0[j]= self.std0[j]**0.5
            self.std1[j]=self.std1[j]/self.countone
            self.std1[j]=self.std1[j]**0.5


    def predict(self,x):
        x=x.T
        n=x.shape[0] #n data points
        k=x.shape[1] #k features
        y_pred=[]
        mean0=self.mean0
        mean1=self.mean1
        std0=self.std0
        std1=self.std1
        for i in range(n):
            prob0=1
            prob1=1
            for j in range(k):
                z0=(x[i][j]-mean0[j])**2/2*(std0[j]**2)
                z1=(x[i][j]-mean1[j])**2/2*(std1[j]**2)
                Pxj_given_0=(1/(std0[j]*(np.sqrt(2*np.pi))*(np.exp(-z0))))
                Pxj_given_1=(1/(std1[j]*(np.sqrt(2*np.pi))*(np.exp(-z1))))
                prob0=prob0*Pxj_given_0
                prob1=prob1*Pxj_given_1
            prob0=prob0*(self.countzero/self.n)
            prob1=prob1*(self.countone/self.n)

            if prob0<prob1:
                y_pred.append(0)
            else:
                y_pred.append(1)

        y_pred=np.array(y_pred)
        y_pred=y_pred.reshape((1,n))
        return y_pred

    def validate(self,y_pred,y_test):
        true_p=true_n=false_n=false_p=0
        n=len(y_pred[0])
        for i in range(n):
            if y_pred[0][i]==y_test[0][i]:
                if y_pred[0][i]==1:
                    true_p+=1
                else:
                    true_n+=1
            else:
                if y_pred[0][i]==1:
                    false_p+=1
                else:
                    false_n+=1
        #precision is nothing but the total number of correct positive predictions /all predicted positive
        precision=(true_p)/(true_p+false_p)
        #recall is equal to number of correct positive predictions / number of actual positives
        recall=(true_p)/(true_p+false_n)
        #harmonic mean of recall and precision
        f1score=(2*precision*recall)/(recall+precision)
        print("----------------Confusion Matrix----------------")
        print("total number of observations:",n)
        print("true positive:",true_p)
        print("true negative:",true_n)
        print("false positive:",false_p)
        print("false negative:",false_n)
        print("--------------------------------------------------")
        print("precision of your model is:",precision)
        print("recall of your model is:",recall)
        print("f1score of your model is:",f1score)
        print("accuracy of your model is:",(true_p+true_n)/n)
```

Training and testing the model with all the columns

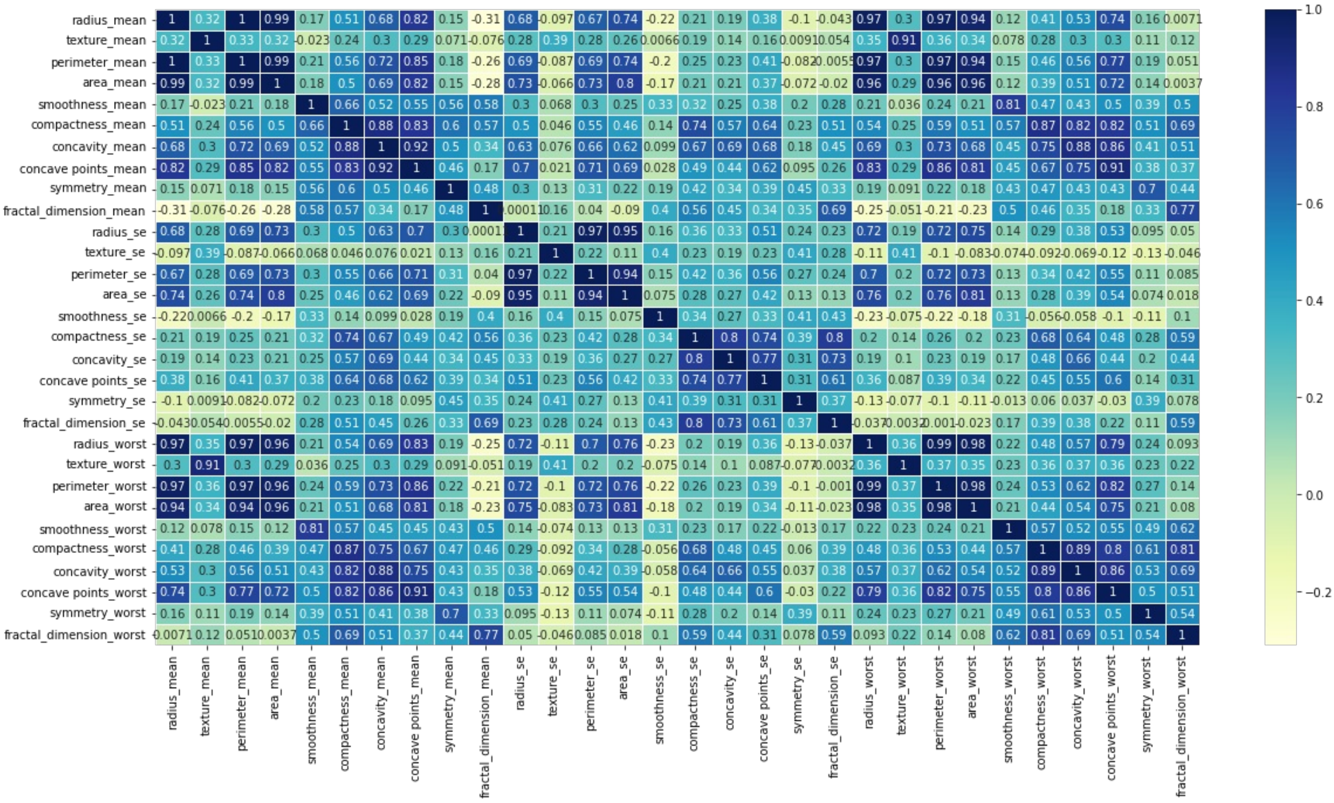
In [21]:
```python
k=binaryNaiveBayes()
k.fit(x_train,y_train)
y_pred=k.predict(x_test)
y_pred.shape
k.validate(y_pred,y_test)
```

```
----------------Confusion Matrix----------------
total number of observations: 171
true positive: 67
true negative: 80
false positive: 22
false negative: 2
-------------------------------------------------
precision of your model is: 0.7528089887640449
recall of your model is: 0.9710144927536232
f1score of your model is: 0.8481012658227848
accuracy of your model is: 0.8596491228070176
```

As we can see , f1 score and accuracy of the model is not good . This maybe due to the fact that many of the featuers are co-related and donot follow the Naive Assumption.

In [22]:

```
import seaborn as sns
cor=d.corr()
plt.figure(figsize=(20,10))
dataplot=sns.heatmap(cor,cmap="YlGnBu",annot=True,linewidths=0.5)
```



Data visualization of all the probability distribution function generated for all the features.

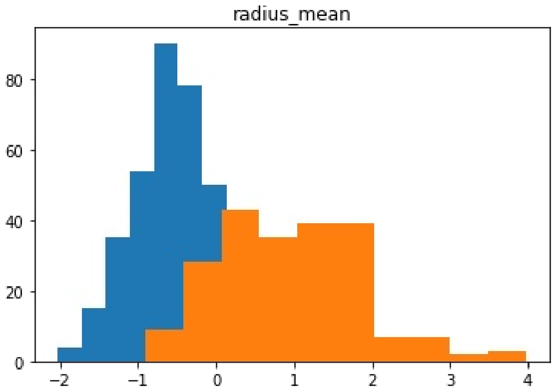
In [23]:

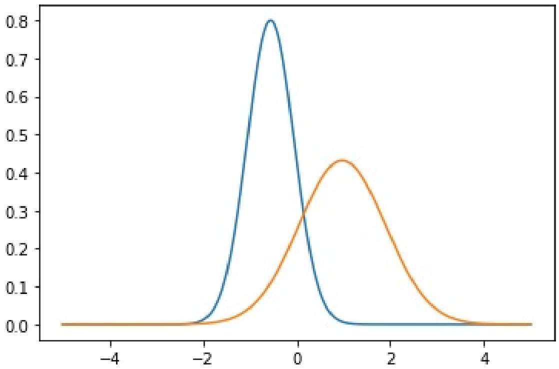
```
from scipy.stats import norm

def visualize():
    d0 = d[d['diagnosis']==0]
    d1 = d[d['diagnosis']==1]
    t=0
    for i in d.columns:
        if i=='diagnosis':
            continue
        else:
            d0[i]
            plt.figure()
            plt.hist(d0[i],bins=10)
            plt.title(d0[i].name)
            plt.hist(d1[i],bins=10)
            plt.figure()
            x = np.arange(-5,5, 0.001)
            plt.plot(x, norm.pdf(x, k.mean0[t], k.std0[t]))
            plt.plot(x, norm.pdf(x, k.mean1[t], k.std1[t]))
            t=t+1
```

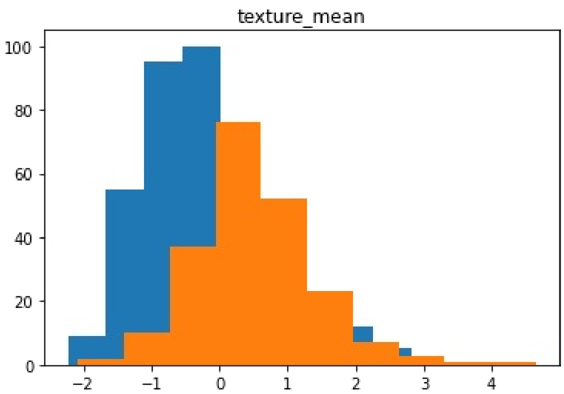
In [24]:

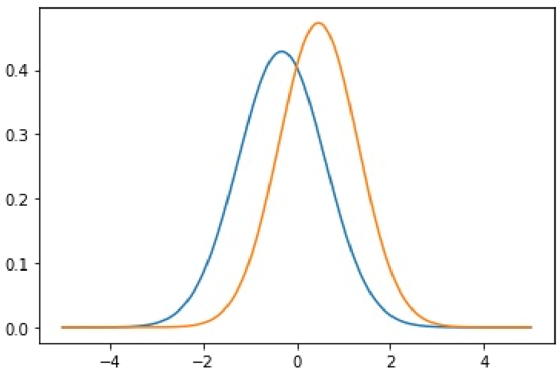
```
visualize() # to visualize the gaussian curves fitted to each classs for all the features in above code
```

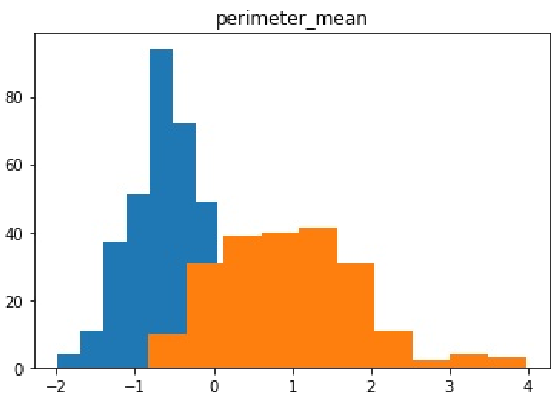
```
C:\Users\aditya\AppData\Local\Temp\ipykernel_5500\1074915522.py:12: RuntimeWarning: More than 20 figures have b
een opened. Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are retained until explic
itly closed and may consume too much memory. (To control this warning, see the rcParam `figure.max_open_warning
`).
  plt.figure()
```

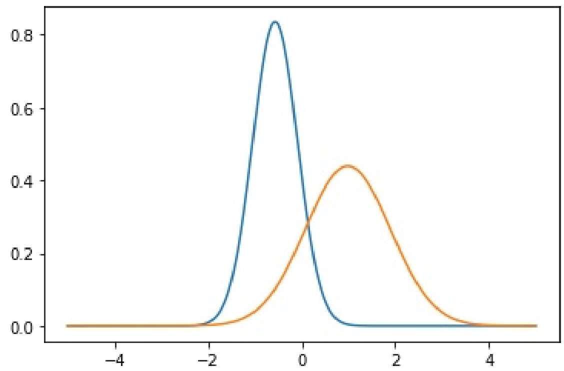

radius_mean

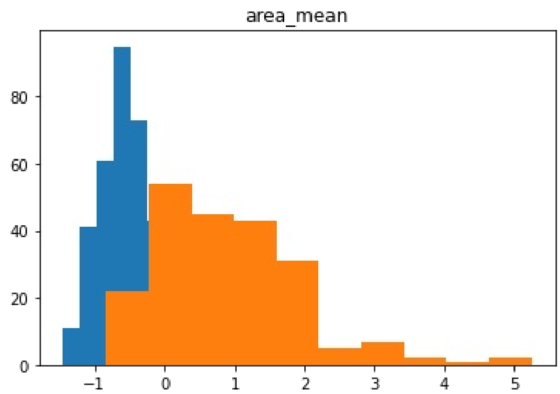


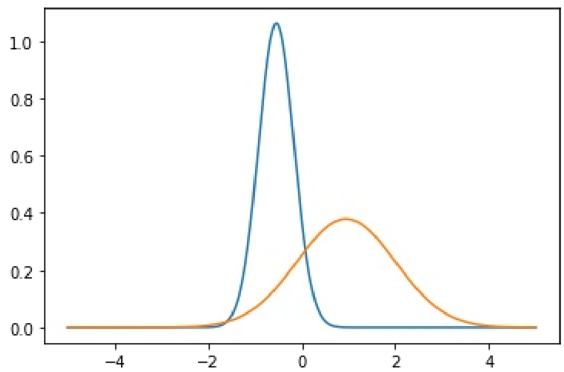

texture_mean

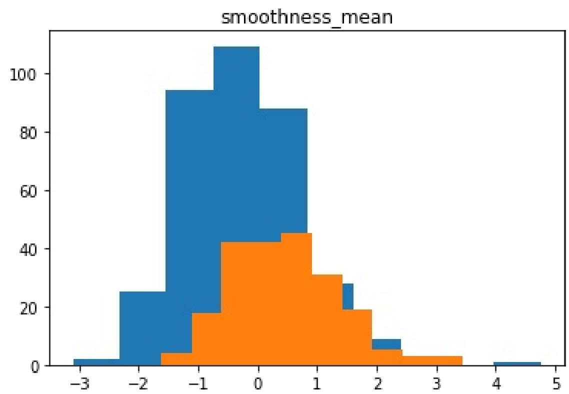


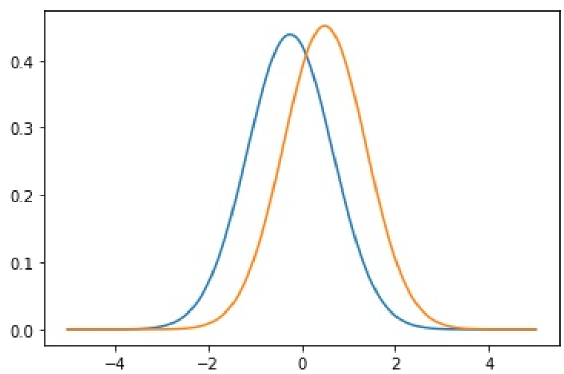

perimeter_mean

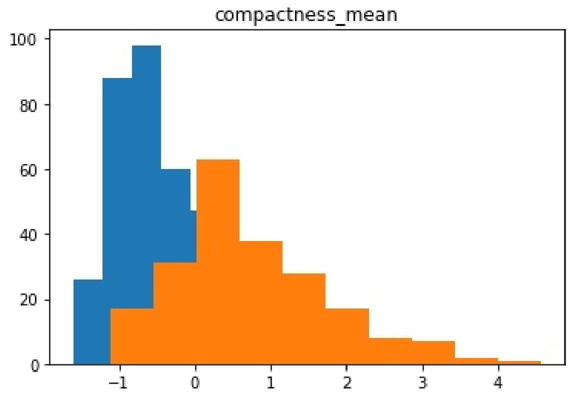


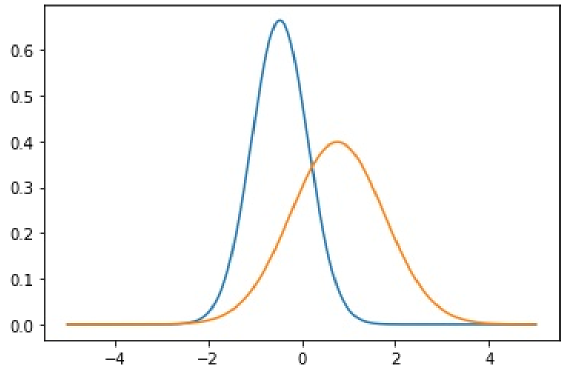

area_mean

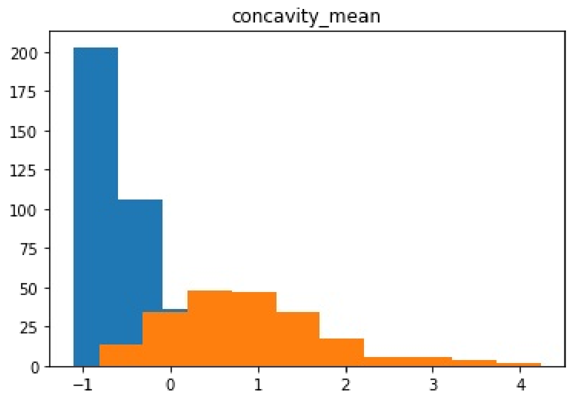


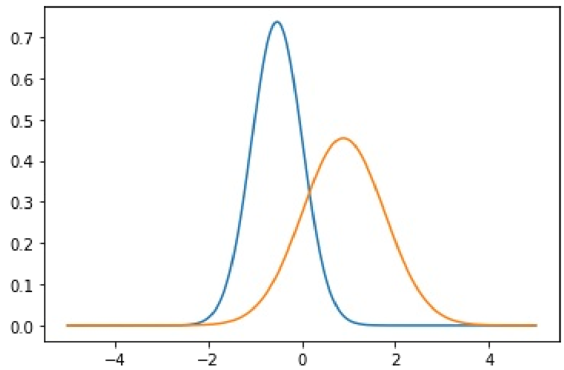

smoothness_mean

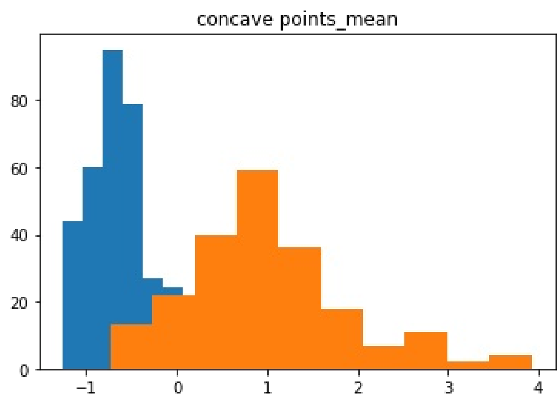


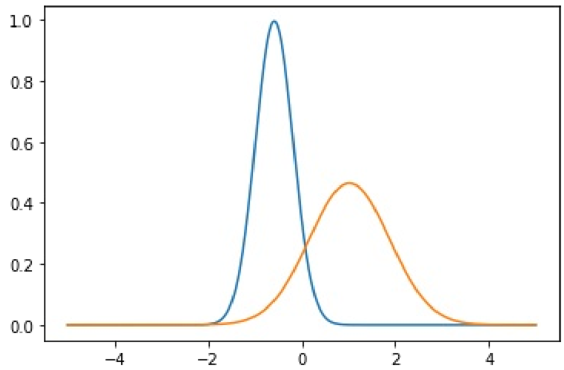

compactness_mean
0
20
40
60
80
100
−1
0
1
2
3
4


0.0
0.1
0.2
0.3
0.4
0.5
0.6
−4
−2
0
2
4

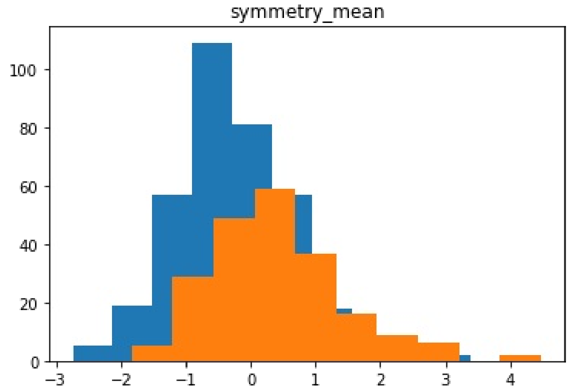

concavity_mean
0
25
50
75
100
125
150
175
200
−1
0
1
2
3
4

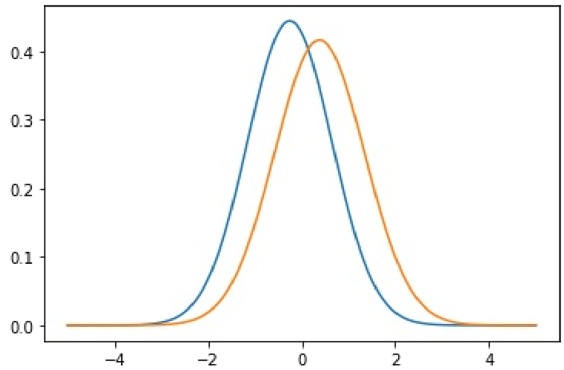

0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
−4
−2
0
2
4


concave points_mean
0
20
40
60
80
−1
0
1
2
3
4

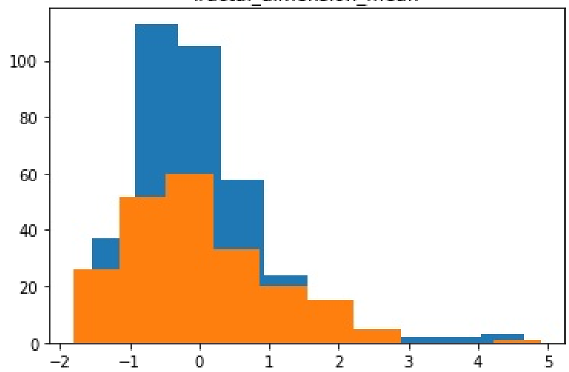


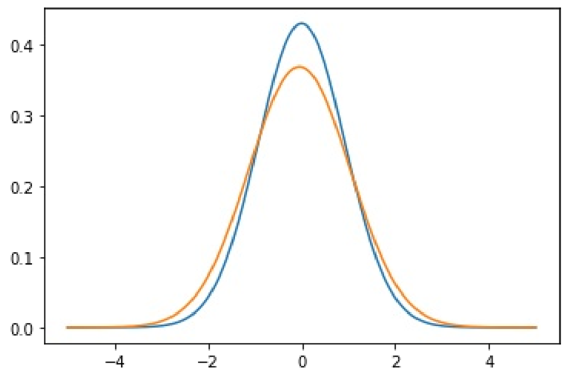
symmetry_mean

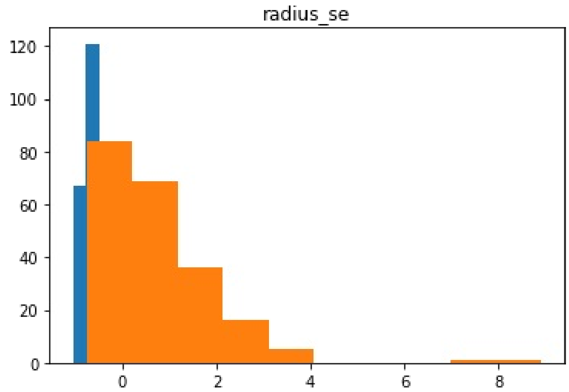


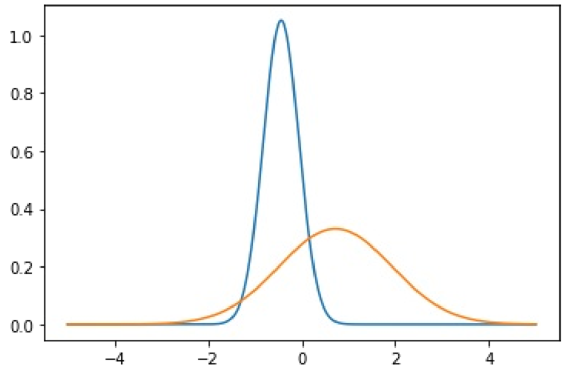


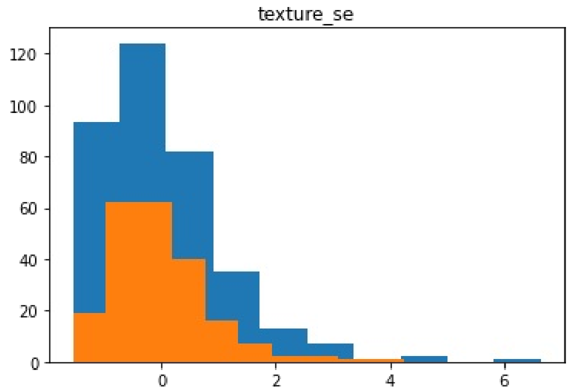
fractal_dimension_mean

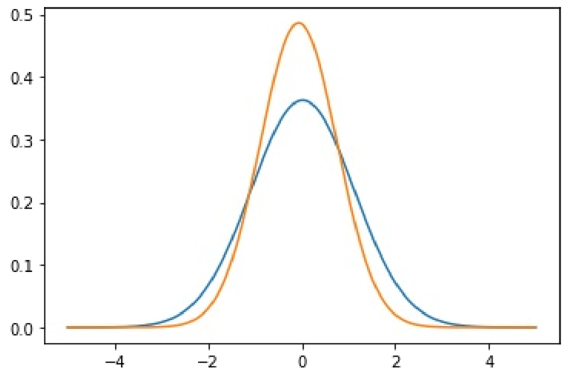


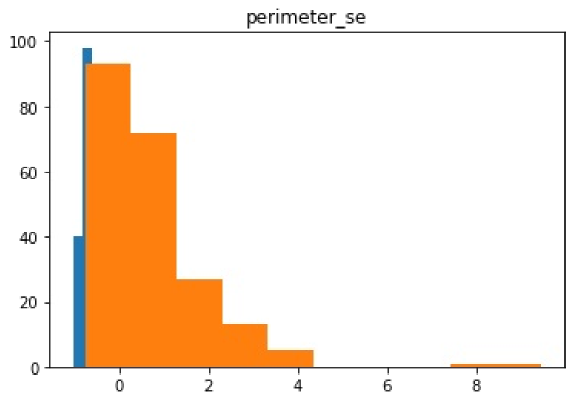


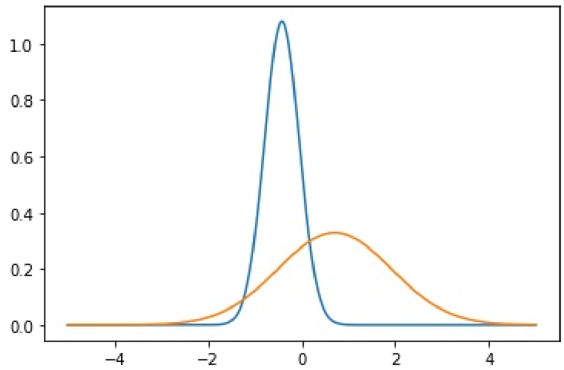

radius_se
0 20 40 60 80 100 120
0 2 4 6 8

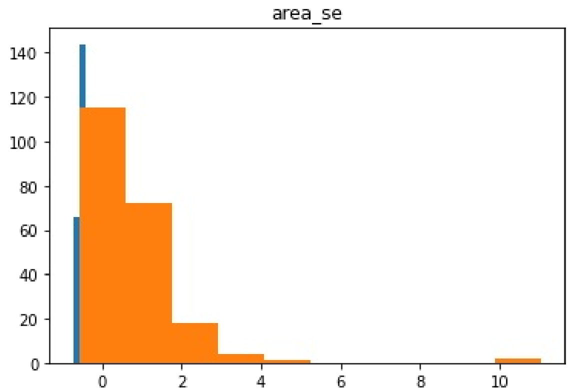

0.0 0.2 0.4 0.6 0.8 1.0
−4 −2 0 2 4

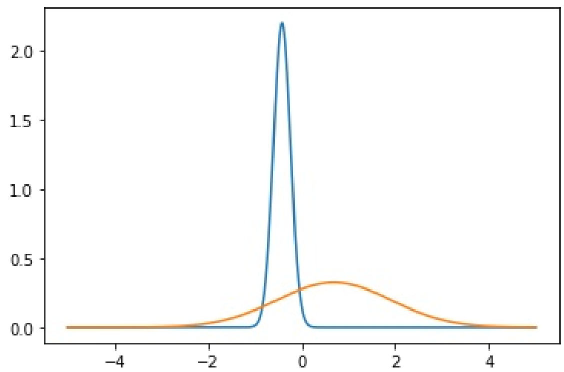

texture_se
0 20 40 60 80 100 120
0 2 4 6

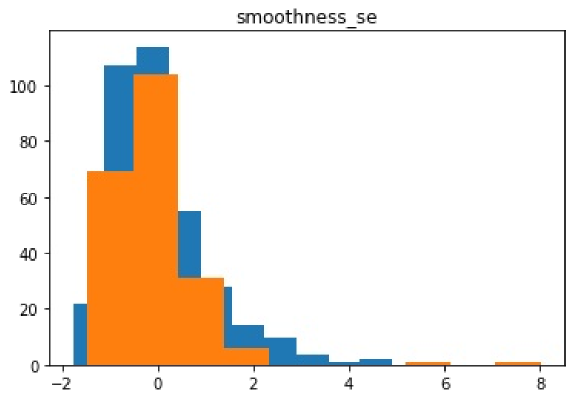

0.0 0.1 0.2 0.3 0.4 0.5
−4 −2 0 2 4

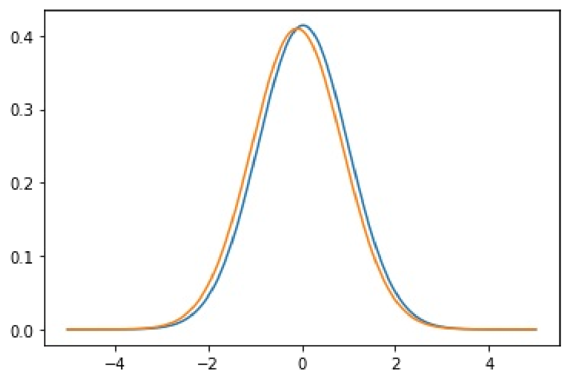

perimeter_se
0 20 40 60 80 100
0 2 4 6 8

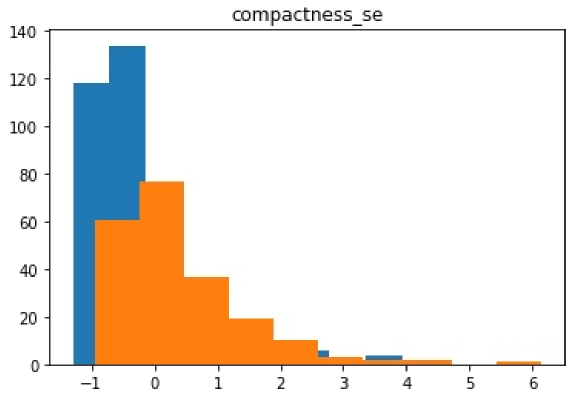

1.0
0.8
0.6
0.4
0.2
0.0
−4 −2 0 2 4

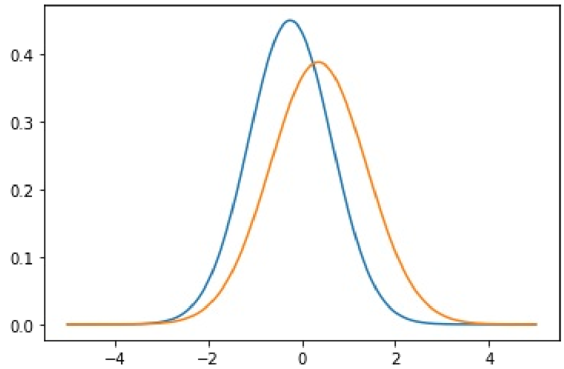

area_se
140
120
100
80
60
40
20
0
0 2 4 6 8 10

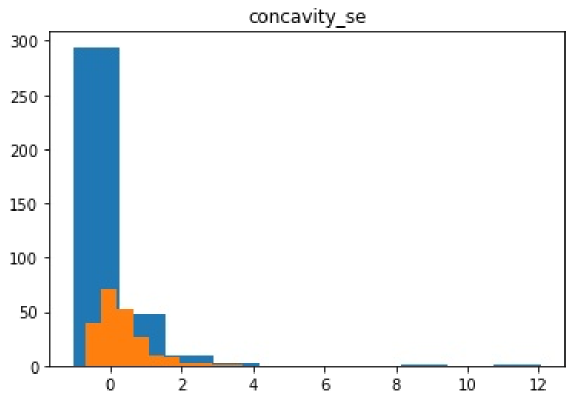

2.0
1.5
1.0
0.5
0.0
−4 −2 0 2 4

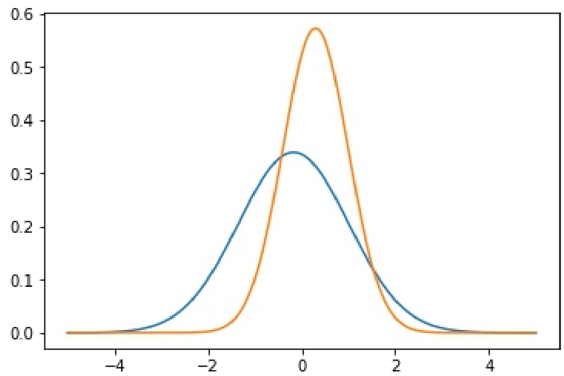

smoothness_se
100
80
60
40
20
0
−2 0 2 4 6 8

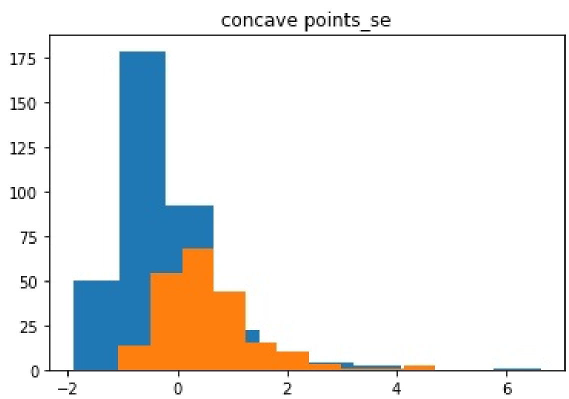

0.4
0.3
0.2
0.1
0.0
−4 −2 0 2 4

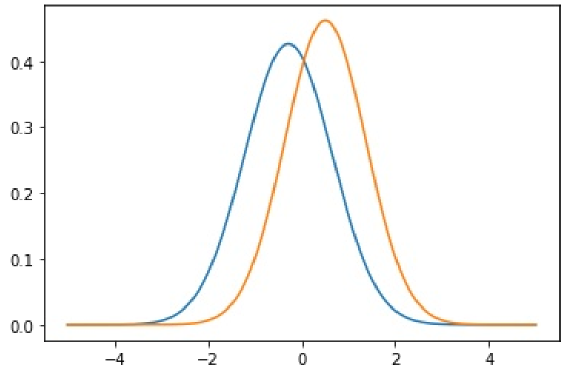

compactness_se



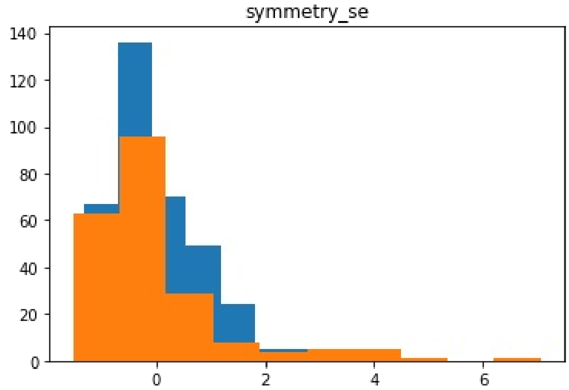

concavity_se

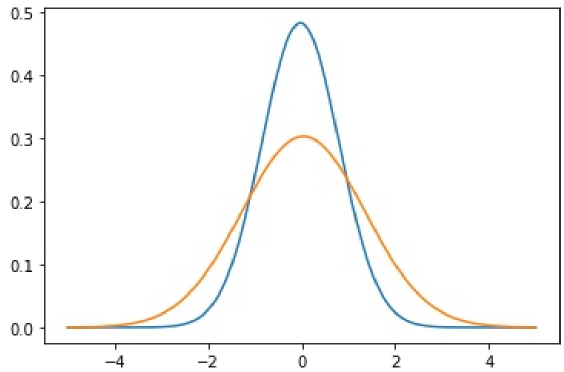



concave points_se

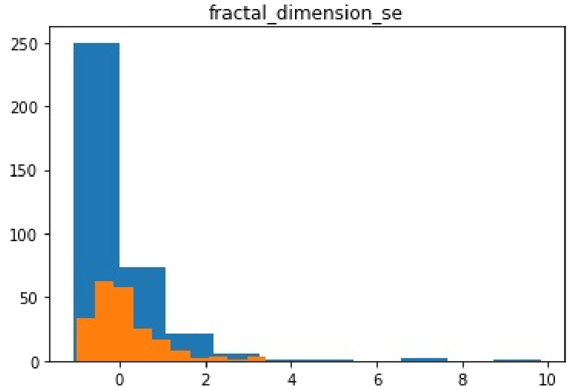


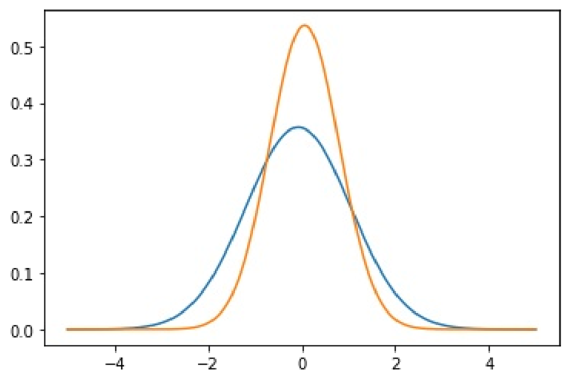
symmetry_se

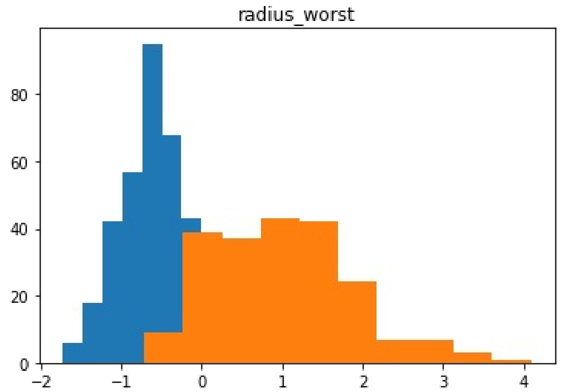


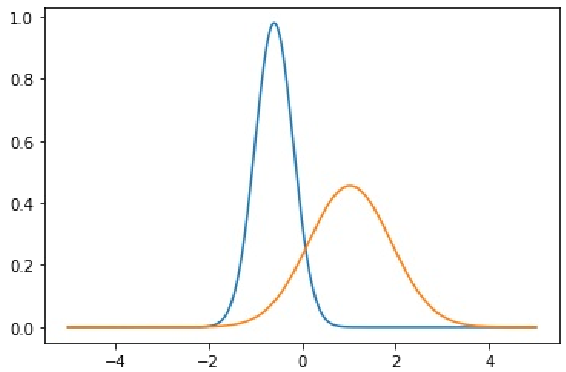


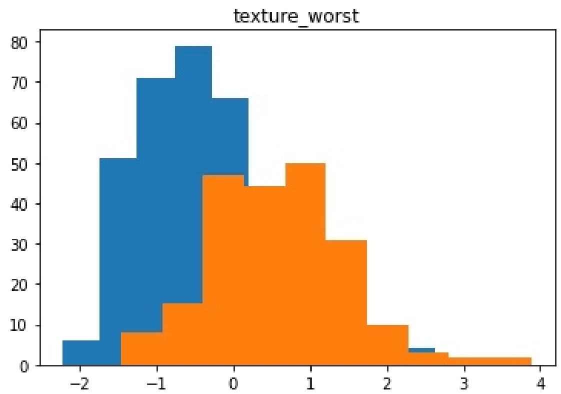
fractal_dimension_se

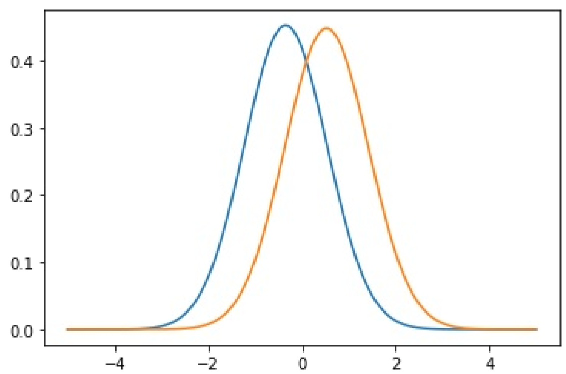


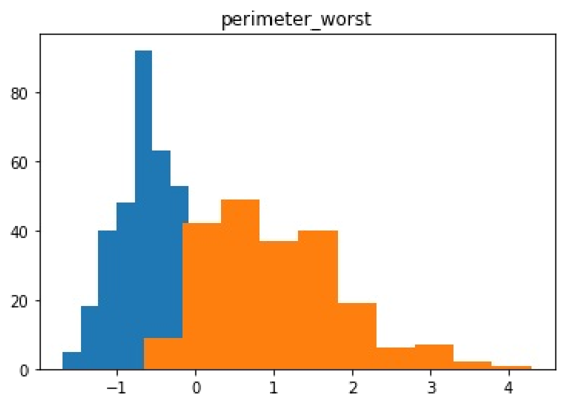


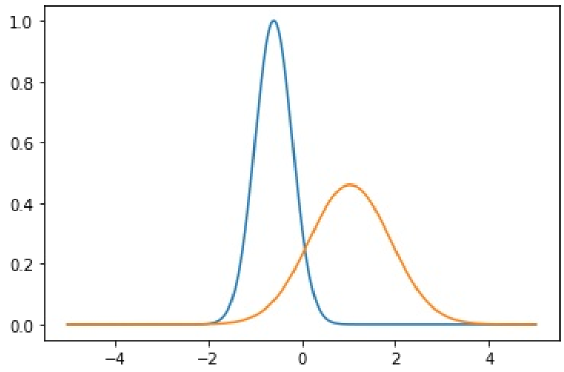

radius_worst



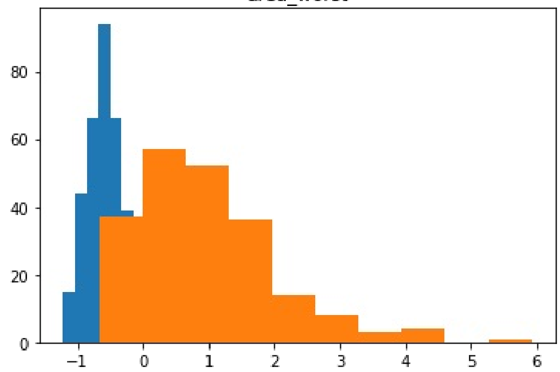

texture_worst

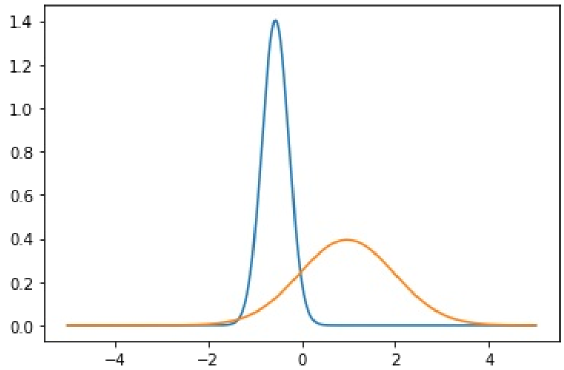



perimeter_worst

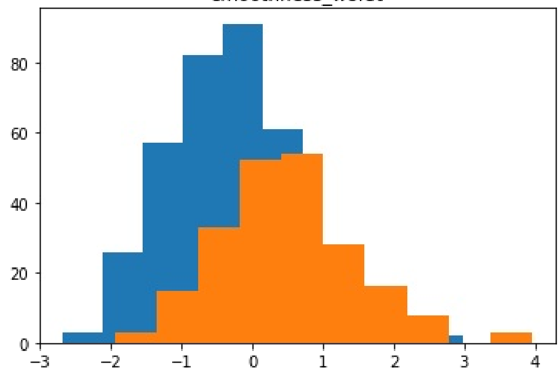


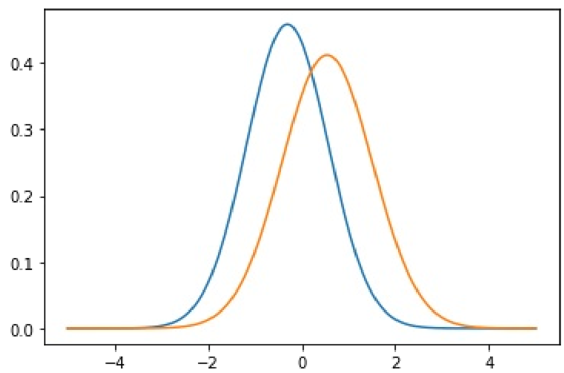
area_worst

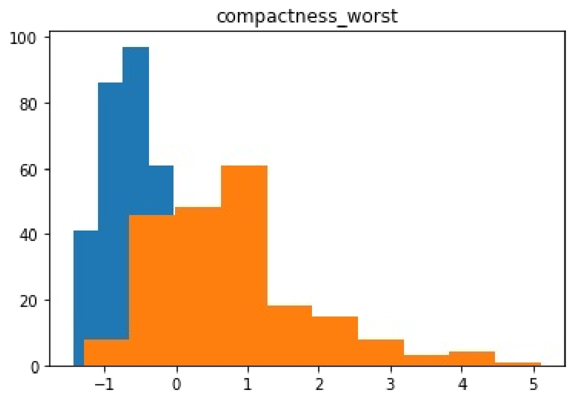


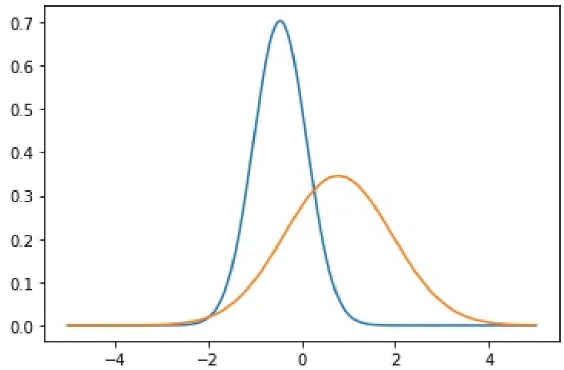


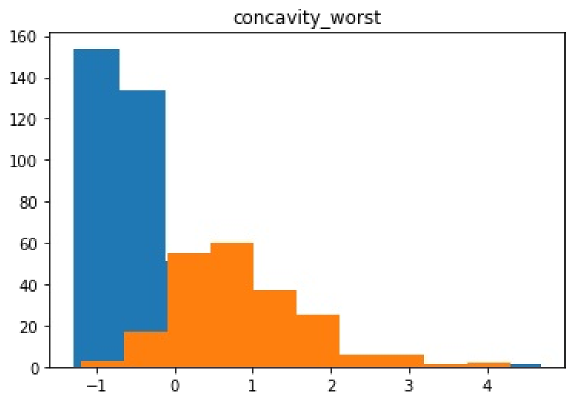
smoothness_worst

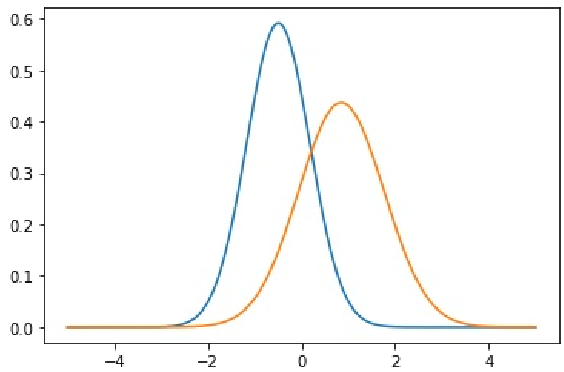


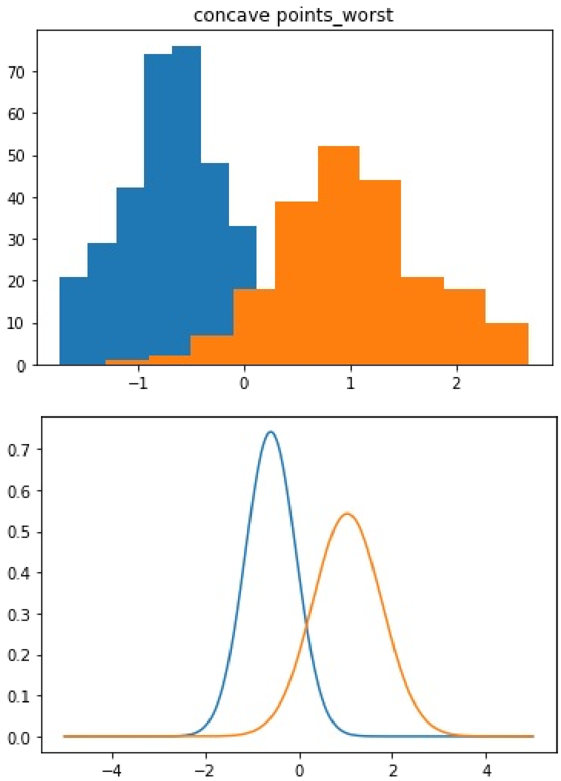


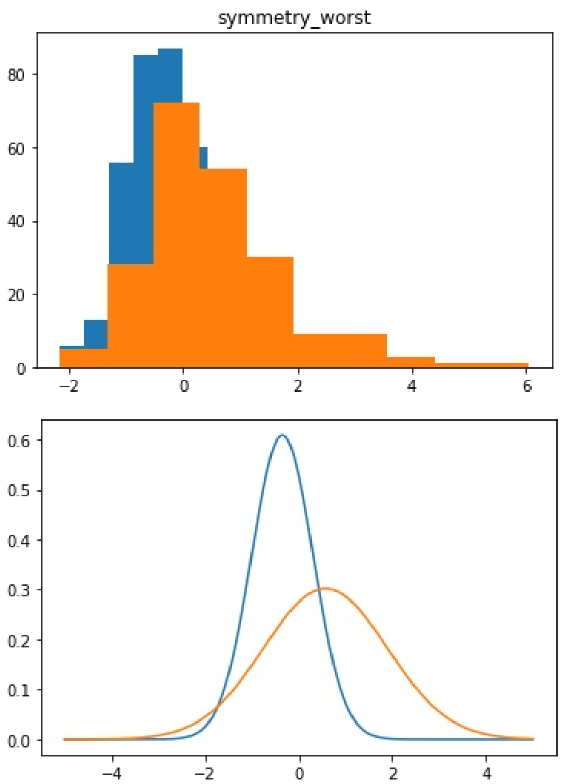

compactness_worst
100
80
60
40
20
0
−1
0
1
2
3
4
5

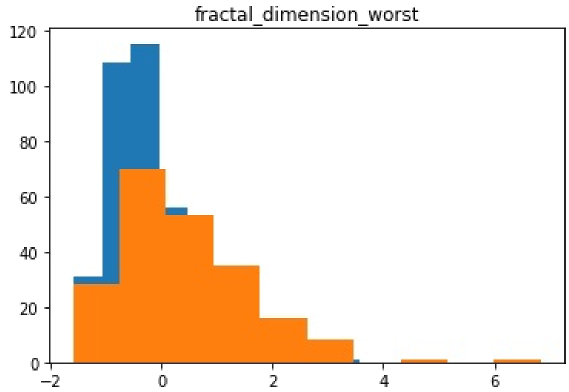

0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−4
−2
0
2
4

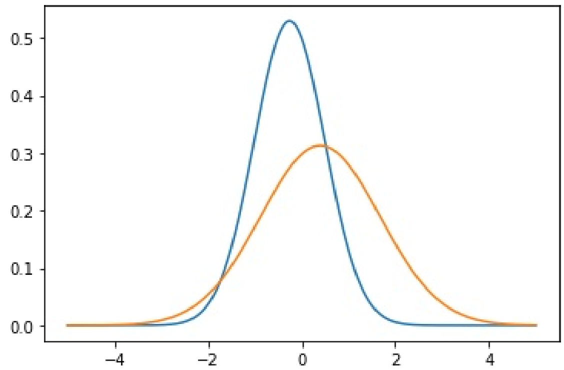

concavity_worst
160
140
120
100
80
60
40
20
0
−1
0
1
2
3
4


0.6
0.5
0.4
0.3
0.2
0.1
0.0
−4
−2
0
2
4


concave points_worst


symmetry_worst





Selecting features

```
In [25]: y=d.iloc[:,0]
         x=d[['radius_mean','texture_mean','perimeter_mean','area_mean','smoothness_mean']]
         x=x.values
         y=y.values
```

```
In [26]: x_test,x_train,y_test,y_train=split(x,y,0.7)
         x_test=np.array(x_test)
         x_train=np.array(x_train)
         y_test=np.array(y_test)
         y_train=np.array(y_train)
```

```
In [27]: x_test=x_test.T
         x_train=x_train.T
         y_test=y_test.reshape(1,x_test.shape[1])
         y_train=y_train.reshape(1,x_train.shape[1])
```

```
In [28]: k=binaryNaiveBayes()
         k.fit(x_train,y_train)
         y_pred=k.predict(x_test)
         y_pred.shape
         k.validate(y_pred,y_test)
```

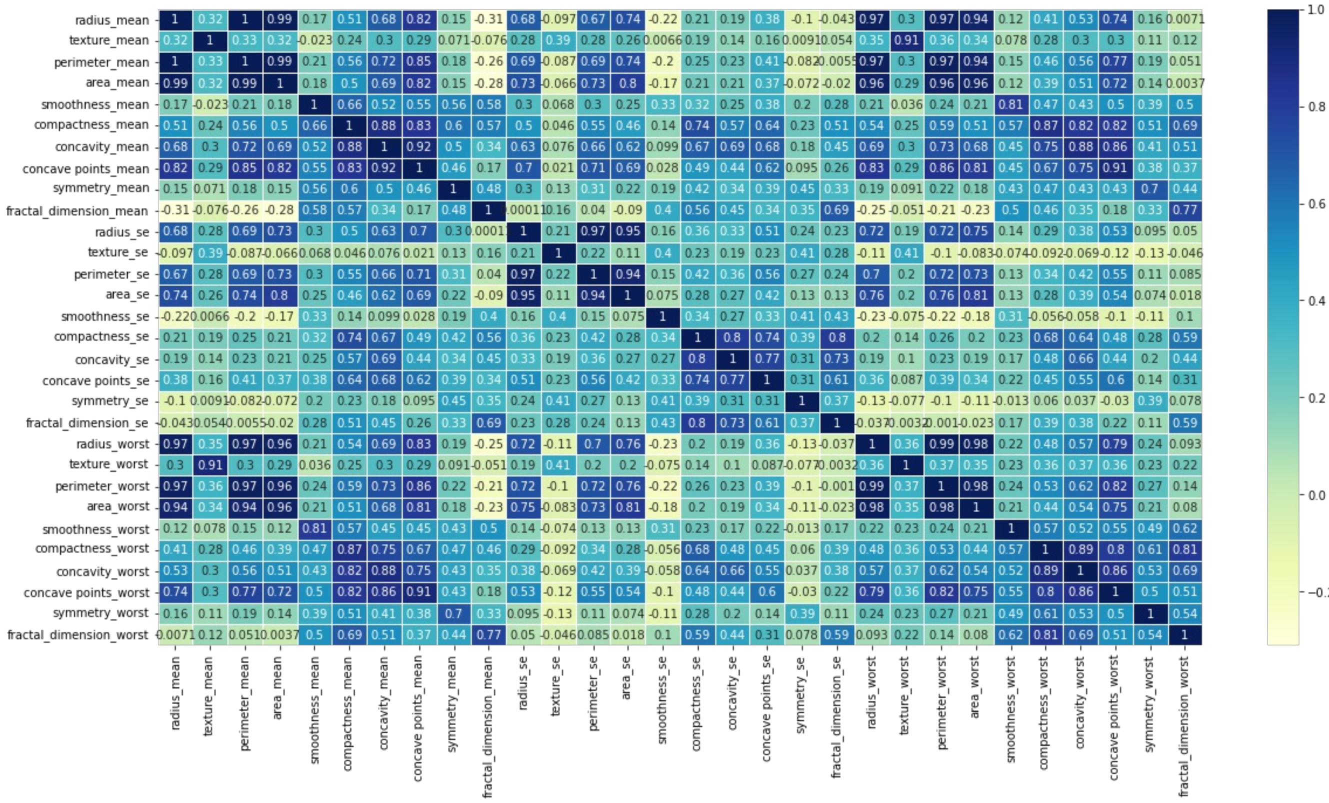
```
----------------Confusion Matrix----------------
total number of observations: 171
true positive: 61
true negative: 78
false positive: 29
false negative: 3
-------------------------------------------------
precision of your model is: 0.6777777777777778
recall of your model is: 0.953125
f1score of your model is: 0.7922077922077922
accuracy of your model is: 0.8128654970760234
```

Even after selecting few features which are not corelated we are not able to imporove the f1 score this maybe due to the fact that the co-related featues might have some more useful information which needs to be considered for classification . Hence multidimensional Gaussian has to be fitted

Selecting features for Multi-dimensional Gaussian

```
In [29]: import seaborn as sns
```

```python
cor=d.corr()
plt.figure(figsize=(20,10))
dataplot=sns.heatmap(cor,cmap="YlGnBu",annot=True,linewidths=0.5)
```



perimeter_mean and area_mean has almost 100% corelation with radius_mean , hence drop it.

From remaining features create a multi dimensioanl gaussian for the features that have more than 0.75 corelation .

Treat the remaining features as independent and fit a 2-D gaussian.

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js