```
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.metrics import accuracy_score, confusion_matrix

df = pd.read_csv('diabetes.csv')
df.head()
```

	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

+ Code + Text

```
temp = df.groupby("Outcome").size()
temp
```

```
Outcome
0    500
1    268
dtype: int64
```

```
y = df['Outcome']
# drop the col 'outcome'
x = df.drop(['Outcome'],axis=1)

# Split data into train & test
x_train, x_test, y_train, y_test = train_test_split(x, y, stratify=y, random_state=42)


x_train.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
count	576.000000	576.000000	576.000000	576.000000	576.000000	576.000000	576.000000	576.000000
mean	3.831597	120.767361	69.170139	20.723958	77.899306	32.064583	0.480200	33.536458
std	3.312864	31.771380	18.699887	15.877307	107.415003	7.861032	0.333188	11.878752
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.084000	21.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.600000	0.245750	24.000000
50%	3.000000	116.500000	72.000000	23.000000	40.000000	32.400000	0.384000	30.000000
75%	6.000000	141.000000	80.000000	32.000000	129.250000	36.525000	0.646250	41.000000
max	17.000000	199.000000	122.000000	99.000000	744.000000	67.100000	2.329000	81.000000

x_test.describe()

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
count	192.000000	192.000000	192.000000	192.000000	192.000000	192.000000	192.000000	192.000000
mean	3.885417	121.276042	68.911458	19.973958	85.500000	31.776562	0.446906	32.354167
std	3.542915	32.650006	21.253333	16.203689	136.216758	7.969892	0.325265	11.381513
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000
25%	1.000000	99.000000	64.000000	0.000000	0.000000	26.075000	0.237000	23.000000
50%	3.000000	119.500000	72.000000	22.000000	0.000000	31.600000	0.343000	28.000000
75%	6.000000	140.000000	80.000000	32.000000	120.000000	36.600000	0.563500	39.000000
max	15.000000	197.000000	106.000000	63.000000	846.000000	59.400000	2.420000	68.000000

```
train_mean_pos = x_train[y_train==1].mean()
train_std_pos = x_train[y_train==1].std()
train_mean_neg = x_train[y_train==0].mean()
train_std_neg = x_train[y_train==0].std()

from math import sqrt
from math import pi
from math import exp
# formula of Gausian NB
def cond_probability(x, mean, std):
  exponent = exp(-((x - mean)**2/(2*std**2)))
  return (1 / (sqrt(2*pi)*std)) * exponent

def predict(row):
  prob_pos = len(x_train[y_train==1]) / len(x_train)
  for i in range(0,len(row)):
    prob_pos = prob_pos * cond_probability(row[i],train_mean_pos[i],train_std_pos[i])
  prob_neg = len(x_train[y_train==0]) / len(x_train)
  for i in range(0,len(row)):
    prob_neg = prob_neg * cond_probability(row[i],train_mean_neg[i],train_std_neg[i])
  return [prob_pos,prob_neg]

predictions_raw = []
for row in x_test.values.tolist():
  predictions_raw.append(predict(row))
```

predictions_raw[0]

```
[1.6299028206157718e-14, 1.0044068228290291e-14]
```

predictions_raw

```
[[1.6299028206157718e-14, 1.0044068228290291e-14],
 [4.1416091656376674e-13, 1.3042909855968897e-12],
 [1.1228918162921918e-13, 3.698652043170301e-12],
 [4.270084880967 41e-13, 7.034723756517606e-13],
 [7.913590093643907e-14, 1.281328360925972e-15],
 [6.035090353659 65e-14, 1.390772897035608e-12],
 [1.6545496022765486e-13, 8.00346760473016e-13],
 [3.1403211509634367e-17, 4.768433856559617e-16],
 [1.270299259926624e-13, 1.1016259499091278e-13],
 [2.1748571686748456e-14, 2.1739905042451717e-12],
 [4.971297069213275e-13, 4.668622177374758e-12],
 [4.5137523313178456e-14, 2.6278537816854723e-15],
 [5.544956140320912e-14, 4.4363696298950895e-13],
 [8.3984990507346 31e-14, 2.789658783732221e-12],
 [2.3296901261242477e-14, 1.4306214129931 64e-14],
 [1.5705424247395808e-16, 2.3912684922010 95e-18],
 [1.3372550935227474e-16, 8.091703856113028e-17],
 [3.7850981610406474e-14, 4.424312795466002e-13],
 [5.1879097868291456e-18, 8.258115874425753e-17],
 [1.2095736900610271e-13, 4.805126942734823e-13],
 [9.658216711483473e-15, 6.459669635048539e-13],
 [1.1831223955213394e-14, 9.596716117647106e-14],
 [6.764511366588288e-14, 3.615779409361753e-12],
 [1.7555770779266 44e-13, 3.0019973779133 35e-14],
 [9.397709330573138e-15, 9.325669412062575e-17],
 [1.1181370060758 26e-14, 3.3419770461683 77e-16],
 [7.687344900125402e-14, 1.5324955999168888e-12],
 [4.6537294420847004e-14, 2.1821689009236796e-12],
 [2.8420836238274412e-14, 1.3687460385561877e-16],
 [1.0336716098079303e-14, 5.7243498652778 55e-16],
 [3.064263619029307e-14, 1.8343172325155 79e-14],
 [7.740869466236125e-22, 8.576082902615116e-20],
 [3.2799869031042725e-14, 1.4346246040234 98e-12],
 [3.3583316981093224e-15, 8.0745545431021 54e-16],
 [1.096277615483422e-15, 1.6209337376166807e-17],
 [1.2650619209498 01e-13, 2.7700385695182056e-12],
 [6.292688660660292e-13, 3.2138378418203097e-12],
 [3.9629147975623396e-14, 2.6453648309256 72e-13],
 [1.1577540063741118e-13, 1.9908086752005 53e-12],
 [6.1420349707482 63e-14, 3.0670513854630 74e-12],
 [8.4583798534465e-14, 2.4626376484433194e-14],
 [2.4508208415499863e-15, 2.4795250600079 09e-13],
 [7.711713709363677e-14, 1.7602849122722366e-12],
 [8.1100125989191 84e-14, 3.9462184831609 15e-14],
 [2.2521228861482883e-16, 3.6750051465924 38e-17],
 [1.5899868755432368e-14, 7.6122267729258 74e-15],
 [2.7929933729708 47e-13, 2.3526306593919896e-14],
 [2.8279469834299 31e-14, 2.0880790640945582e-13],
 [1.2995520917319937e-14, 1.5868715069105584e-15],
 [5.6691077341895 06e-23, 9.6005897714041 65e-28],
 [8.3282263574744 12e-14, 3.3605345306489253e-13],
 [2.5569442519390 17e-15, 7.1041612340742 14e-16],
 [3.6398165988425 4e-13, 7.9571418270228 34e-13],
 [2.6488598289024356e-16, 6.2139946709797 65e-16],
 [1.1010202863161147e-14, 3.8489365673927 42e-15],
 [3.4525035812643664e-14, 1.9768802759399186e-14],
 [5.9819162665062 76e-14, 6.7851469204571354e-15],
 [1.6399845908843516e-13, 2.0510188018021236e-12],
```

```
predictions = []
for row in predictions_raw:
  if(row[0]>row[1]):
    predictions.append(1)
  else:
    predictions.append(0)

# comparing our predictions and actual output for accuracy
accuracy_score(y_test.tolist(),predictions)
```

```
0.7135416666666666
```

```
# plotting the output for comparison
confusion_matrix(y_test.tolist(),predictions)
```

```
array([[96, 29],
       [26, 41]])
```

```
model = GaussianNB() # Gaussian NB has been used becuase the data is continuous
model.fit(x_train,y_train)
```

```
GaussianNB()
```

```
confusion_matrix(y_test,model.predict(x_test))
```

```
array([[96, 29],
       [26, 41]])
```

✓ 0s completed at 15:06