Importing Libraries

In [1]:
```
import pandas as pd
import numpy as np
from numpy import log,dot,exp,shape
import matplotlib.pyplot as plt
import seaborn as sns
```

Importing the Dataset

In [2]:
```
dataset=pd.read_csv("diabetes.csv")
```

In [3]:
```
dataset
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	
764	2	122	70	27	0	36.8	
765	5	121	72	23	112	26.2	
766	1	126	60	0	0	30.1	
767	1	93	70	31	0	30.4	

768 rows × 9 columns

In [4]:
```
dataset.describe()
```

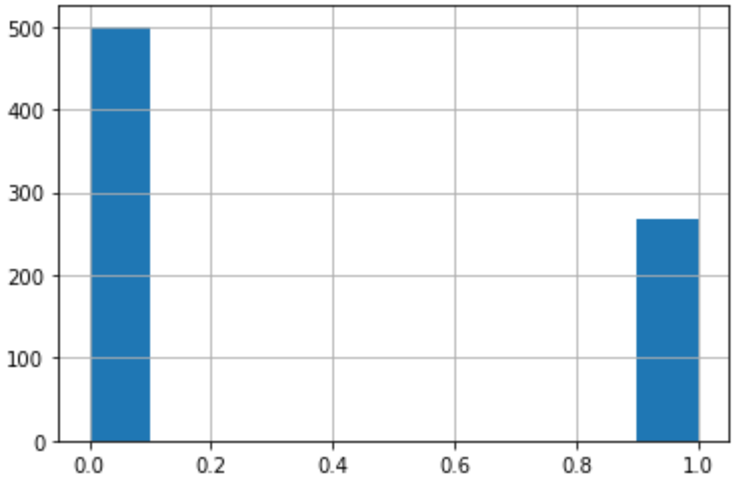
Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diab
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

Univariate Naive Bayes

In [5]:
```
dataset["Outcome"].hist()
```

Out[5]: <AxesSubplot:>



Analyzing the coorelation matrix in order to choose our feature

In [6]:
```
dataset.corr()
```

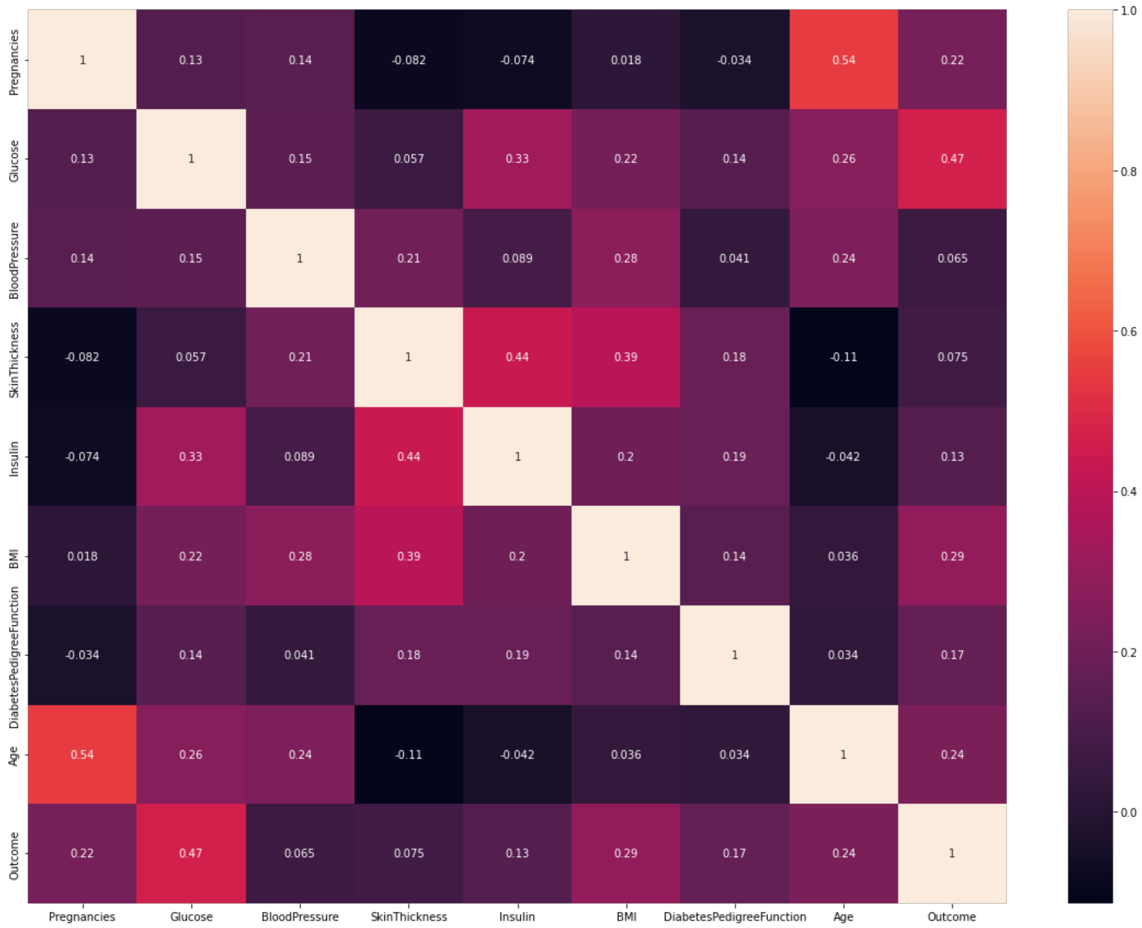
Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	(
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	(
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	(
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	(
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	(
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	(
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	(
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	(

In [7]:

```
plt.figure(figsize = (20, 15))
corr = dataset.corr()
sns.heatmap(corr,
        xticklabels=corr.columns,
        yticklabels=corr.columns,
           annot=True)
```

Out[7]: <AxesSubplot:>



As from the above table we can see that The feature Glucode is best correlated with our target variable (Outcome) , therefore choosing our univariate feature to be Glucose

In [8]:
```
dataset=dataset[["Glucose","Outcome"]]
dataset
```

Out[8]:

	Glucose	Outcome
0	148	1
1	85	0
2	183	1
3	89	0
4	137	1
...	...	...
763	101	0
764	122	0
765	121	0
766	126	1
767	93	0

768 rows × 2 columns

Setting the feature and target variable

In [9]:
```
x=dataset.iloc[:,:-1].values
y=dataset.iloc[:,-1].values
```

In [10]:
```
x
```

Out[10]:
```
array([[148],
       [ 85],
       [183],
       [ 89],
       [137],
       [116],
       [ 78],
       [115],
       [197],
       [125],
       [110],
       [168],
       [139],
       [189],
       [166],
       [100],
       [118],
       [107],
       [103],
```

In [11]:
```
y
```

Out[11]:
```
array([1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1,
0, 0,
       1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0,
0, 1,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
1, 0,
       1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
0, 1,
       1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1,
1, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0,
       1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0,
0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1,
0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0,
1, 1,
       1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0,
0, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1,
0, 0,
       1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1,
1, 0,
       0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0,
1, 0,
       1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0,
1, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0,
       0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0,
0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0,
1, 0,
       0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1,
0, 1,
       0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
0, 0,
       1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0,
0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0,
0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0,
1, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0,
1, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
```

```
0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0, 1,
       0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
0, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1,
1, 0,
       0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0,
0, 0,
       0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0,
1, 0,
       1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0])
```

Filling the missing values

In [12]:
```
from sklearn.impute import SimpleImputer
imputer=SimpleImputer(missing_values=np.nan,strategy='mean')
imputer.fit(x)
x=imputer.transform(x)
```

In [13]:
```
x
```

Out[13]:
```
array([[148.],
       [ 85.],
       [183.],
       [ 89.],
       [137.],
       [116.],
       [ 78.],
       [115.],
       [197.],
       [125.],
       [110.],
       [168.],
       [139.],
       [189.],
       [166.],
       [100.],
       [118.],
       [107.],
       [103.],
       [115.],
```

Splitting the dataset into training set and test set(test data set 20%)

In [31]:
```
train=dataset.iloc[:616,:]
x_train=x[:616,:]
x_test=x[616:,:]
y_train=y[:616]
y_test=y[616:]
```

In [32]:
```
x_train
```

Out[32]:
```
array([[148.],
       [ 85.],
       [183.],
       [ 89.],
       [137.],
       [116.],
       [ 78.],
       [115.],
       [197.],
       [125.],
       [110.],
       [168.],
       [139.],
       [189.],
       [166.],
       [100.],
       [118.],
       [107.],
       [103.],
```

In [33]:
```
x_test
```

Out[33]:
```
array([[117.],
       [ 68.],
       [112.],
       [119.],
       [112.],
       [ 92.],
       [183.],
       [ 94.],
       [108.],
       [ 90.],
       [125.],
       [132.],
       [128.],
       [ 94.],
       [114.],
       [102.],
       [111.],
       [128.],
       [ 92.],
```

In [34]:
```
y_train
```

Out[34]:
```
array([1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1,
0, 0,
       1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0,
0, 1,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
1, 0,
       1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
0, 1,
       1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1,
1, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0,
       1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0,
0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1,
0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0,
1, 1,
       1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0,
0, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1,
0, 0,
       1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1,
1, 0,
       0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0,
1, 0,
       1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0,
1, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0,
       0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0,
0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0,
1, 0,
       0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1,
0, 1,
       0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
0, 0,
       1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0,
0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0,
0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0,
1, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0,
1, 0])
```

In [35]:
```
y_test
```

Out[35]:
```
array([0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0, 1,
       0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
0, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1,
1, 0,
       0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0,
0, 0,
       0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0,
1, 0,
       1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0])
```

In [36]:
```
len(x_train)+len(x_test)
```

Out[36]: 768

In [37]:
```
len(y_train)+len(y_test)
```

Out[37]: 768

In [38]:
```
len(x_train)
```

Out[38]: 616

In [39]:
```
len(y_train)
```

Out[39]: 616

Feature Scaling

In [40]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.transform(x_test)
```

In [41]:
```
x_train
```

Out[41]:
```
array([[ 0.84404469],
       [-1.08628039],
       [ 1.91644751],
       [-0.96372007],
       [ 0.5070038 ],
       [-0.13643789],
       [-1.30076095],
       [-0.16707797],
       [ 2.34540864],
       [ 0.13932283],
       [-0.32027837],
       [ 1.4568463 ],
       [ 0.56828396],
       [ 2.10028799],
       [ 1.39556614],
       [-0.62667918],
       [-0.07515773],
       [-0.41219862],
       [-0.53475894],
```

In [42]:
```
x_test
```

Out[42]:
```
array([[-0.10579781],
       [-1.60716176],
       [-0.25899821],
       [-0.04451765],
       [-0.25899821],
       [-0.87179983],
       [ 1.91644751],
       [-0.81051966],
       [-0.38155854],
       [-0.93307999],
       [ 0.13932283],
       [ 0.3538034 ],
       [ 0.23124308],
       [-0.81051966],
       [-0.19771805],
       [-0.56539902],
       [-0.28963829],
       [ 0.23124308],
       [-0.87179983],
```

Defining Naive Bayes Classification

In [56]:
```
def prior_cal(data, y):
    class_set = sorted(list(data[y].unique()))
    prior = []
    for i in class_set:
        prior.append(len(data[data[y]==i])/len(data))
    return prior
```

In [57]:
```
def likelihood_gaussian(data, name, val, y, label):
    feat = list(data.columns)
    data = data[data[y]==label]
    mean, std = data[name].mean(), data[name].std()
    PX_given_Y = (1 / (np.sqrt(2 * np.pi) * std)) *  np.exp(-((val-me
    return PX_given_Y
```

In [58]:
```
def naive_bayes(data, x, y):
    # extracting feature names
    features = list(data.columns)[:-1]

    # calculate prior
    prior = prior_cal(data, y)

    y_pred = []
    # loop over every data sample
    for x in x:
        # calculate likelihood
        labels = sorted(list(data[y].unique()))
        likelihood = [1]*len(labels)
        for j in range(len(labels)):
            for i in range(len(features)):
                likelihood[j] *= likelihood_gaussian(data, features[i

        # calculate posterior probability (numerator only)
        post_prob = [1]*len(labels)
        for j in range(len(labels)):
            post_prob[j] = likelihood[j] * prior[j]

        y_pred.append(np.argmax(post_prob))

    return np.array(y_pred)
```

In [59]:
```
y_pred = naive_bayes(train, x_test, y="Outcome")
```

In [60]:
```
y_pred
```

Out[60]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [68]:
```
y_test
```

Out[68]:
```
array([0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0, 1,
       0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
0, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1,
1, 0,
       0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0,
0, 0,
       0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0,
1, 0,
       1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0])
```

In [70]:
```
from sklearn.metrics import confusion_matrix, f1_score
print(confusion_matrix(y_test, y_pred))
print(f1_score(y_test, y_pred))
```

```
[[98  0]
 [54  0]]
0.0
```

Multivariate Naive Bayes

In [71]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("darkgrid")
```

In [74]:
```
dataset = pd.read_csv("diabetes.csv")
```

In [75]:
```
dataset
```

Out[75]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	
764	2	122	70	27	0	36.8	
765	5	121	72	23	112	26.2	
766	1	126	60	0	0	30.1	
767	1	93	70	31	0	30.4	

768 rows × 9 columns

In [76]:
```
dataset.head()
```

Out[76]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunct
0	6	148	72	35	0	33.6	0.
1	1	85	66	29	0	26.6	0.
2	8	183	64	0	0	23.3	0.
3	1	89	66	23	94	28.1	0.
4	0	137	40	35	168	43.1	2.

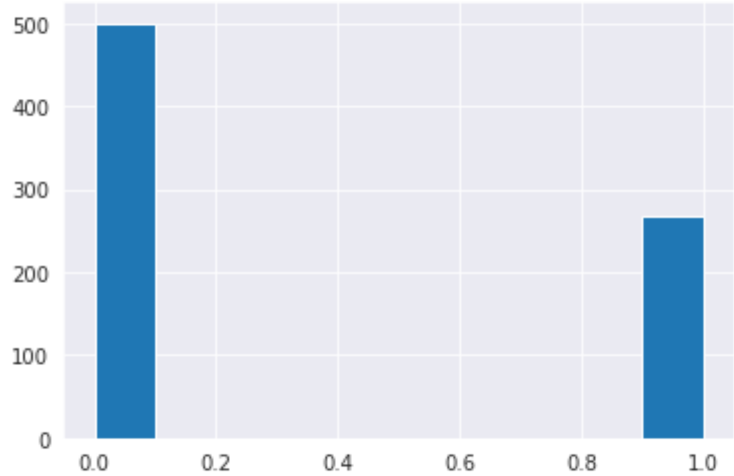
In [77]:
```
dataset.describe()
```

Out[77]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diab
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

In [78]:
```
dataset["Outcome"].hist()
```

Out[78]:
```
<AxesSubplot:>
```



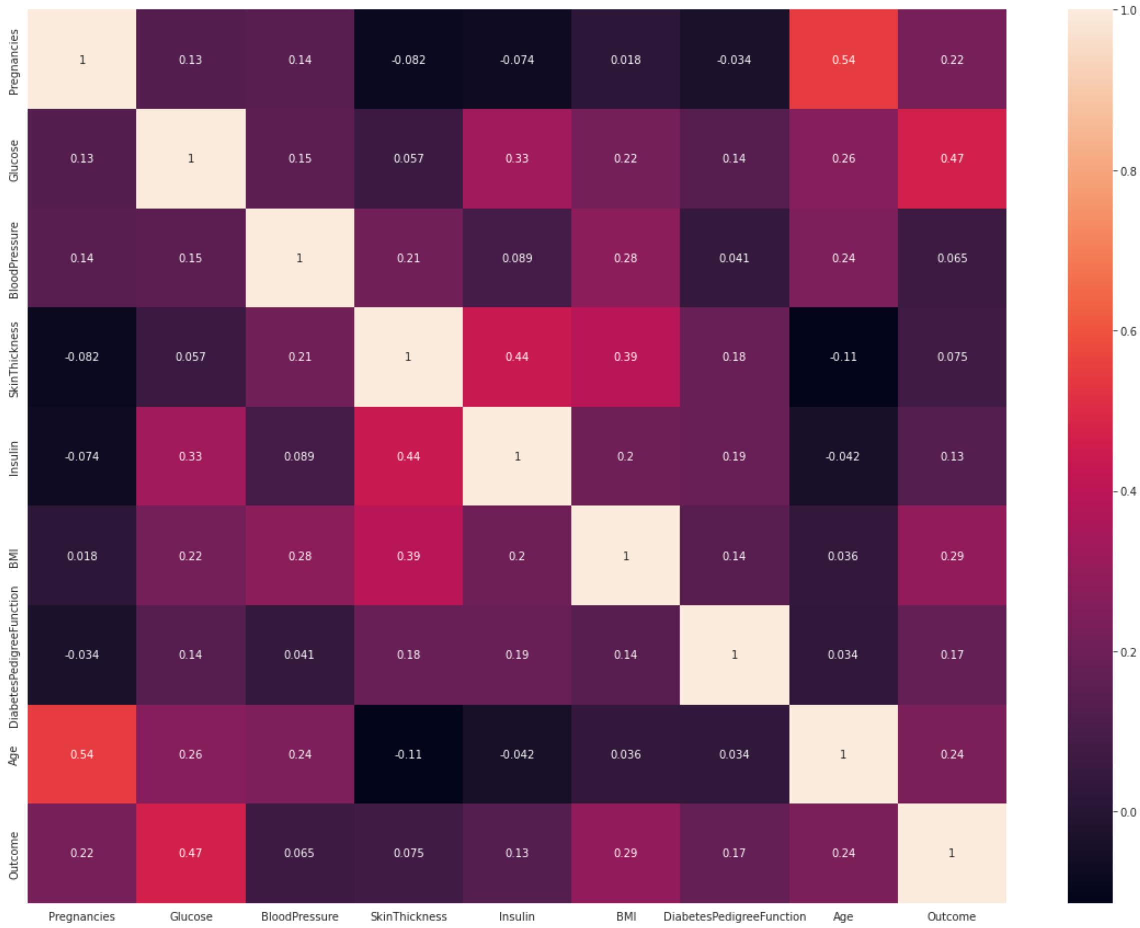
In [79]:
```
dataset.corr()
```

Out[79]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	(
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	(
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	(
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	(
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	(
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	(
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	(
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	(

In [80]:
```
plt.figure(figsize = (20, 15))
corr = dataset.corr()
sns.heatmap(corr,
        xticklabels=corr.columns,
        yticklabels=corr.columns,
           annot=True)
```

Out[80]: <AxesSubplot:>



In [81]:
```
def calculate_prior(df, Y):
    classes = sorted(list(df[Y].unique()))
    prior = []
    for i in classes:
        prior.append(len(df[df[Y]==i])/len(df))
    return prior
```

In [82]:
```
def calculate_likelihood_gaussian(df, feat_name, feat_val, Y, label):
    feat = list(df.columns)
    df = df[df[Y]==label]
    mean, std = df[feat_name].mean(), df[feat_name].std()
    p_x_given_y = (1 / (np.sqrt(2 * np.pi) * std)) *  np.exp(-((feat_
    return p_x_given_y
```

In [83]:
```
def naive_bayes_gaussian(df, X, Y):
    # get feature names
    features = list(df.columns)[:-1]

    # calculate prior
    prior = calculate_prior(df, Y)

    Y_pred = []
    # loop over every data sample
    for x in X:
        # calculate likelihood
        labels = sorted(list(df[Y].unique()))
        likelihood = [1]*len(labels)
        for j in range(len(labels)):
            for i in range(len(features)):
                likelihood[j] *= calculate_likelihood_gaussian(df, fe

        # calculate posterior probability (numerator only)
        post_prob = [1]*len(labels)
        for j in range(len(labels)):
            post_prob[j] = likelihood[j] * prior[j]

        Y_pred.append(np.argmax(post_prob))

    return np.array(Y_pred)
```

In [84]:
```
train_1=dataset.iloc[:616,:]
x1_train=x[:616,:]
x1_test=x[616:,:]
y1_train=y[:616]
y1_test=y[616:]
```

In [85]:
```
train_1
```

Out[85]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	
...	...	...	...	...	...	...	
611	3	174	58	22	194	32.9	
612	7	168	88	42	321	38.2	
613	6	105	80	28	0	32.5	
614	11	138	74	26	144	36.1	
615	3	106	72	0	0	25.8	

616 rows × 9 columns

In [87]:
```
y_pred = naive_bayes_gaussian(train, X=x_test, Y="Outcome")
```

In [88]:
```
y_pred
```

Out[88]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [89]:
```
from sklearn.metrics import confusion_matrix, f1_score
print(confusion_matrix(y_test, y_pred))
print(f1_score(y_test, y_pred))
```

```
[[98  0]
 [54  0]]
0.0
```