Naive Bayes Classifier Model - Pima Indians Diabetes Database

Objective

- The objective is to build Naive Bayes classifier model which can diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset.

Naive Bayes Classifier Algorithm

- Naive Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.
- It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.
- The Naive Bayes algorithm is comprised of Naive and Bayes. It is called Naive because it assumes that the occurrence of a certain feature is independent of the occurrence of other features and it is called Bayes because it depends on the principle of Bayes' Theorem.

Bayes' Theorem & its Formula

- Bayes theorem is a theorem in probability and statistics, named after the Reverend Thomas Bayes, that helps in determining the probability of an event that is based on some event that has already occurred. It depends on the conditional probability.
- The formula for Bayes' theorem is given as: P(A|B)=P(B|A)P(A) / P(B)
- Here, P(A) = how likely A happens(Prior knowledge)- The probability of a hypothesis is true before any evidence is present.
- P(B) = how likely B happens(Marginalization)- The probability of observing the evidence.
- P(A|B) = how likely A happens given that B has happened(Posterior)-The probability of a hypothesis is true given the evidence.
- P(B|A) = how likely B happens given that A has happened(Likelihood)- The probability of seeing the evidence if the hypothesis is true.

Types of Naive Bayes classifiers:

- 1. Gaussian Naive Bayes - It makes predictions based on the probability of each possible outcome. It makes a strong assumption that all features are independent of each other, given the class label which means that the algorithm considers each feature individually and assumes that they contribute equally to the probability of the class.
- 2. Multinomial Naive Bayes - It is used for categorizing documents or text into multiple classes. It is a popular algorithm used for spam detection, sentiment analysis, and text classification.
- 3. Bernoulli Naive Bayes - It predicts the probability of a sample belonging to a particular class. It is usedd for binary classification problems, where the target variable can take only two values, usually 0 or 1.

Dataset source & brief

- This dataset is sourced from kaggle & is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. In particular, all patients here are females at least 21 years old of Pima Indian heritage. The dataset has 9 columns including the Traget variable 'Outcome'.

Import Libraries

In [1]:

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
%matplotlib inline

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

Load the dataset

In [2]:

```
dataset=pd.read_csv(r"C:\Users\manme\Documents\Priya\Stats and ML\Dataset\diabetes.csv")
dataset.head()
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc
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.

Check basic information

In [3]:

```
dataset.shape # check shape
```

Out[3]:

```
(768, 9)
```

In [4]:

```
dataset.describe().T.style.background_gradient(cmap='Greens')  #statistical summary
```

Out[4]:

	count	mean	std	min	25%	5
Pregnancies	768.000000	3.845052	3.369578	0.000000	1.000000	3.000(
Glucose	768.000000	120.894531	31.972618	0.000000	99.000000	117.000(
BloodPressure	768.000000	69.105469	19.355807	0.000000	62.000000	72.000(
SkinThickness	768.000000	20.536458	15.952218	0.000000	0.000000	23.000(
Insulin	768.000000	79.799479	115.244002	0.000000	0.000000	30.500(
BMI	768.000000	31.992578	7.884160	0.000000	27.300000	32.000(
DiabetesPedigreeFunction	768.000000	0.471876	0.331329	0.078000	0.243750	0.372(
Age	768.000000	33.240885	11.760232	21.000000	24.000000	29.000(
Outcome	768.000000	0.348958	0.476951	0.000000	0.000000	0.000(

In [5]:

```
dataset.duplicated().sum()  #check duplicates
```

Out[5]:

```
0
```

In [6]:

```
dataset.info()  # check 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 [7]:

```
for i in dataset.columns:
    print("**********************************", i ,
          "*****************************************")
    print()
    print(set(dataset[i].tolist()))
    print()
```

```
********************************** Pregnancies **********************
******************

{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17}

********************************** Glucose ***************************
***************

{0, 44, 56, 57, 61, 62, 65, 67, 68, 71, 72, 73, 74, 75, 76, 77, 78, 79,
80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97,
98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 11
2, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 12
6, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 14
0, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 15
4, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 16
8, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 18
2, 183, 184, 186, 187, 188, 189, 190, 191, 193, 194, 195, 196, 197, 19
8, 199}
```

- '0' is present in Glucose,BloodPressure, SkinThickness, Insulin and BMI which cannot be the case so we need to treat them.

In [8]:

```
df= dataset.copy(deep = True)
```

In [9]:

```
df[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = df[
    ['Glucose','BloodPressure','SkinThickness','Insulin','BMI']].replace(0,np.NaN)
```

In [10]:

```
df.isnull().sum()   # Check missing values after replacing 0 values with nan
```

Out[10]:

```
Pregnancies                 0
Glucose                     5
BloodPressure              35
SkinThickness             227
Insulin                   374
BMI                        11
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [11]:

```
# Handling missing values with median
df['Glucose'].fillna(df['Glucose'].median(), inplace = True)
df['BloodPressure'].fillna(df['BloodPressure'].median(), inplace = True)
df['SkinThickness'].fillna(df['SkinThickness'].median(), inplace = True)
df['Insulin'].fillna(df['Insulin'].median(), inplace = True)
df['BMI'].fillna(df['BMI'].median(), inplace = True)
```

In [12]:

```
df['Outcome'].value_counts() # check balance of data
```

Out[12]:

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

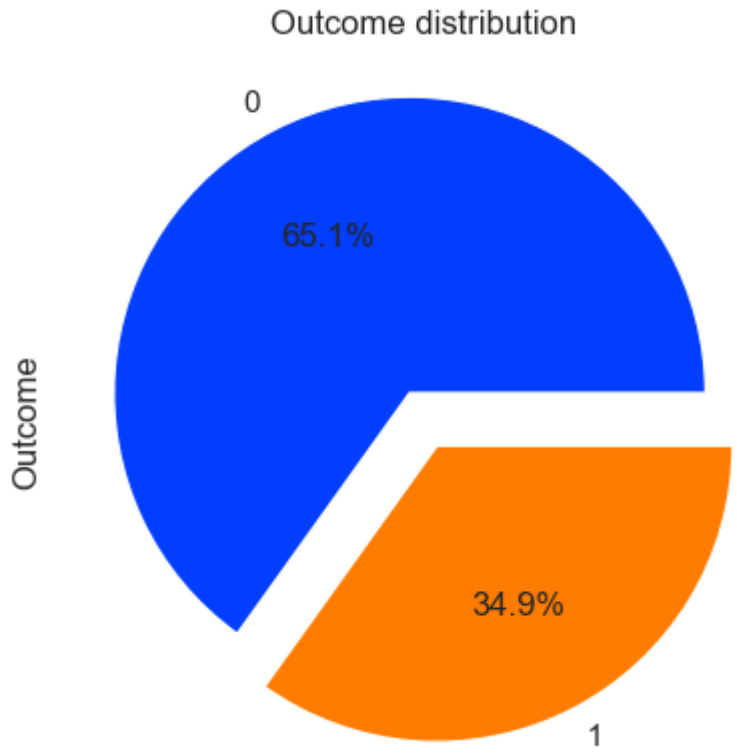
Exploratory Data Analysis

In [13]:

```
sns.set_theme(palette='bright',style='whitegrid')
```

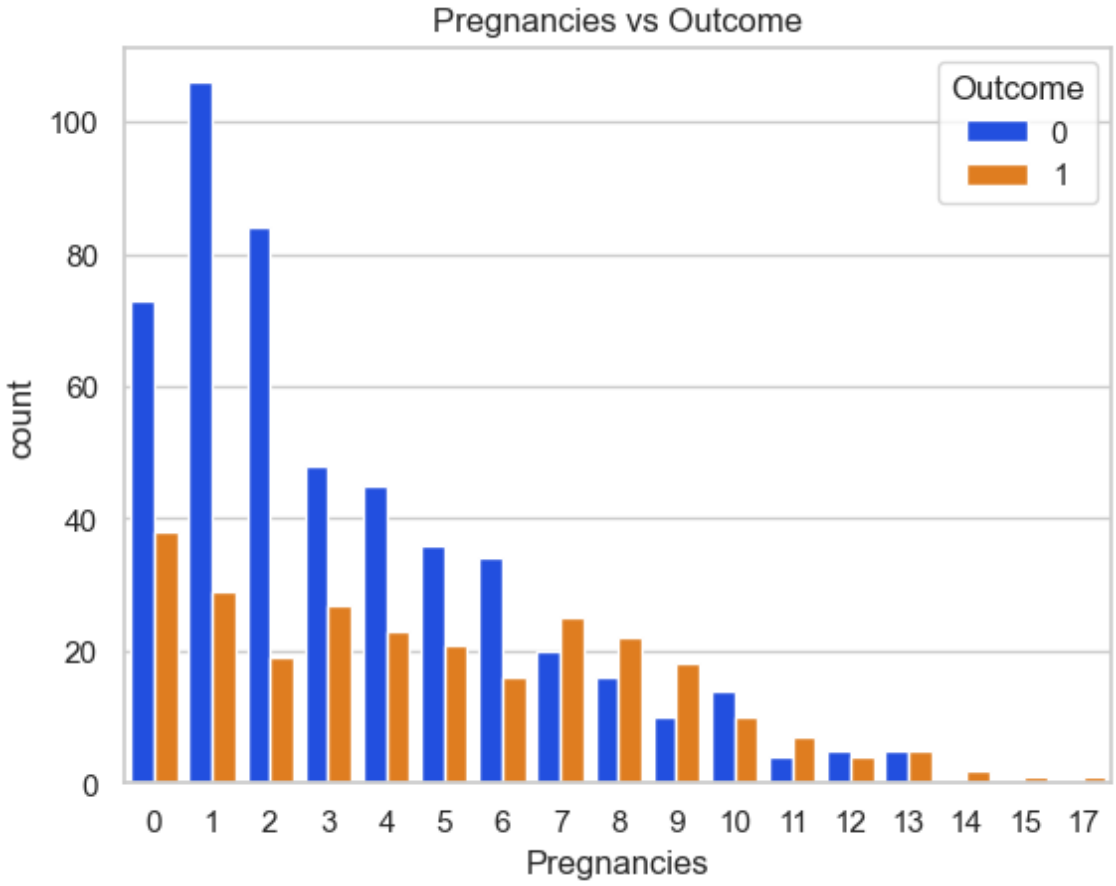
In [14]:

```
df['Outcome'].value_counts().plot(kind='pie',explode=[0.1,0.1],autopct='%0.1f%%')
plt.title('Outcome distribution')
plt.show()
```



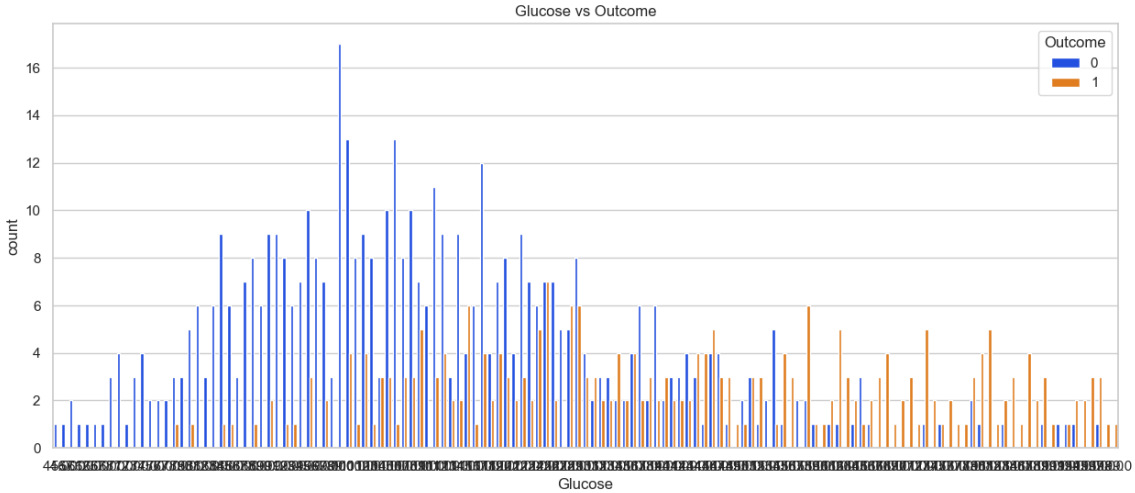
In [15]:

```
sns.countplot(x='Pregnancies',hue='Outcome', data=df)
plt.title('Pregnancies vs Outcome')
plt.show()
```



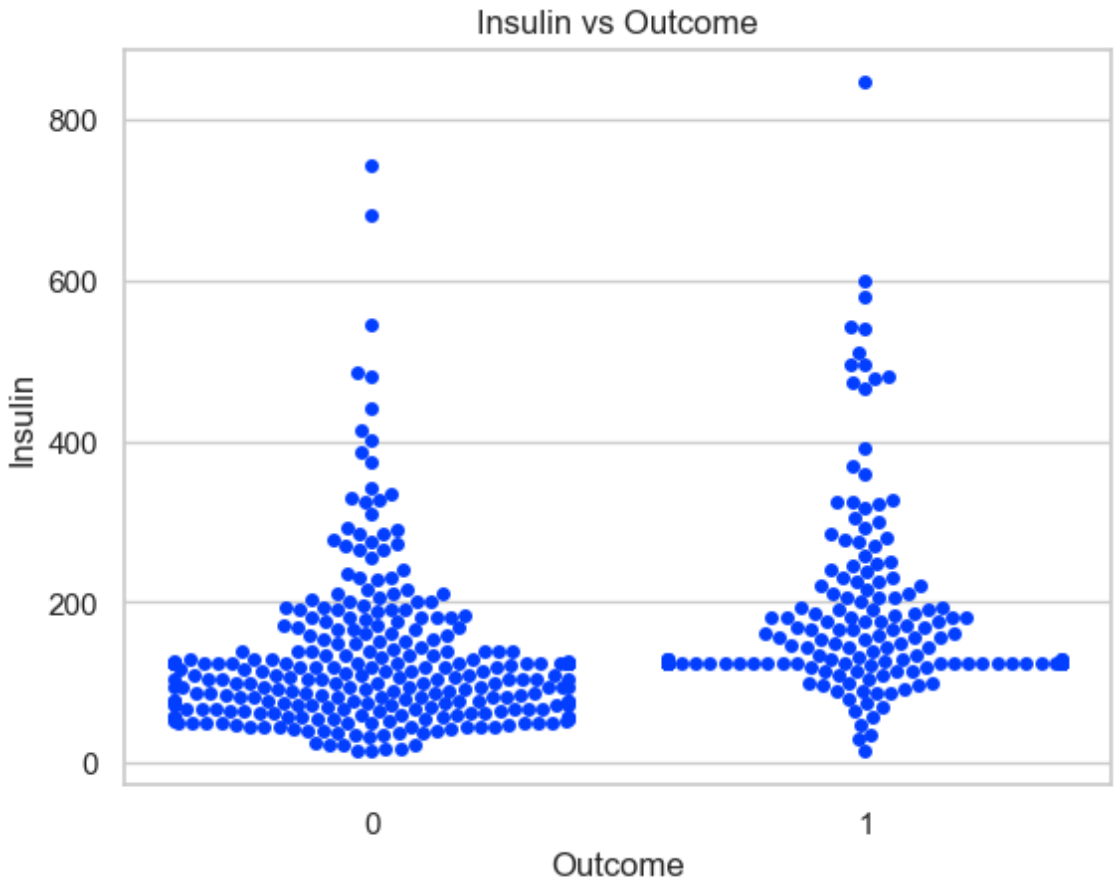
In [16]:

```
plt.figure(figsize=(15,6))
sns.countplot(x = 'Glucose', hue = 'Outcome', data=df)
plt.title('Glucose vs Outcome')
plt.show()
```



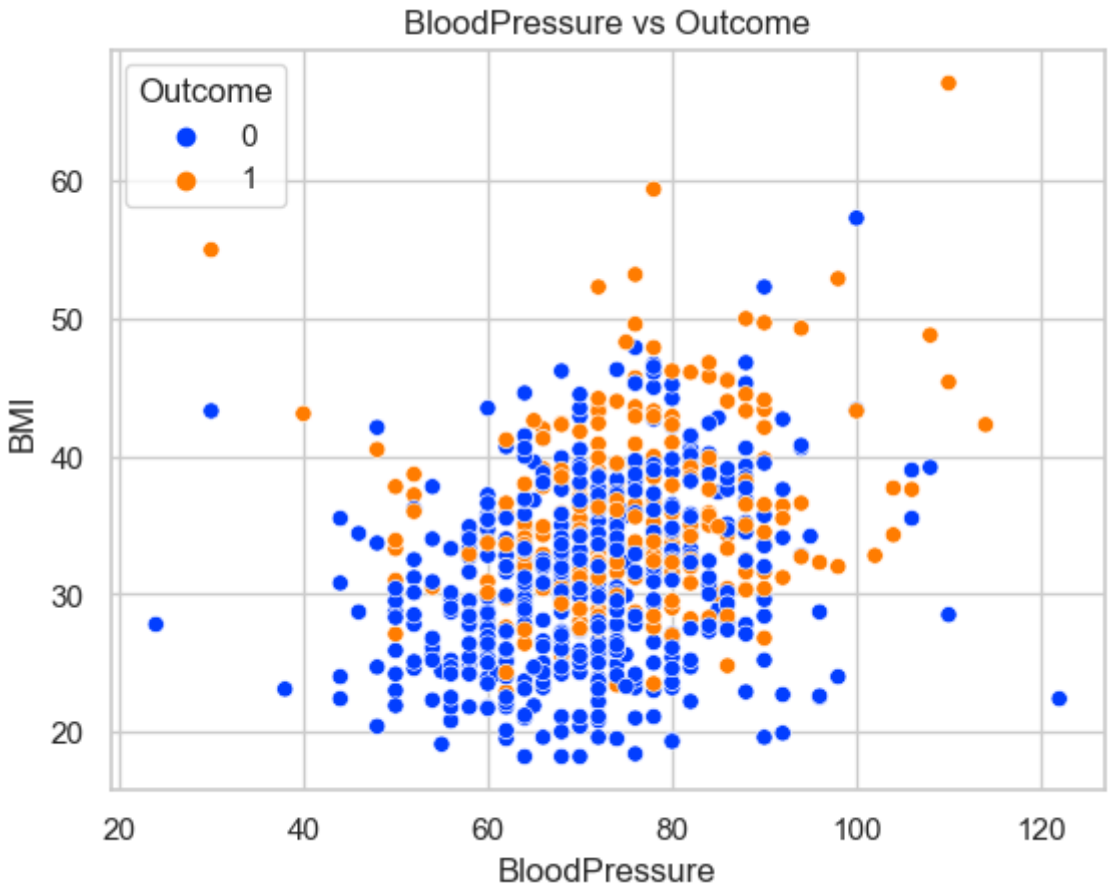
In [17]:

```
sns.swarmplot(y='Insulin',x='Outcome',data=df)
plt.title('Insulin vs Outcome')
plt.show()
```



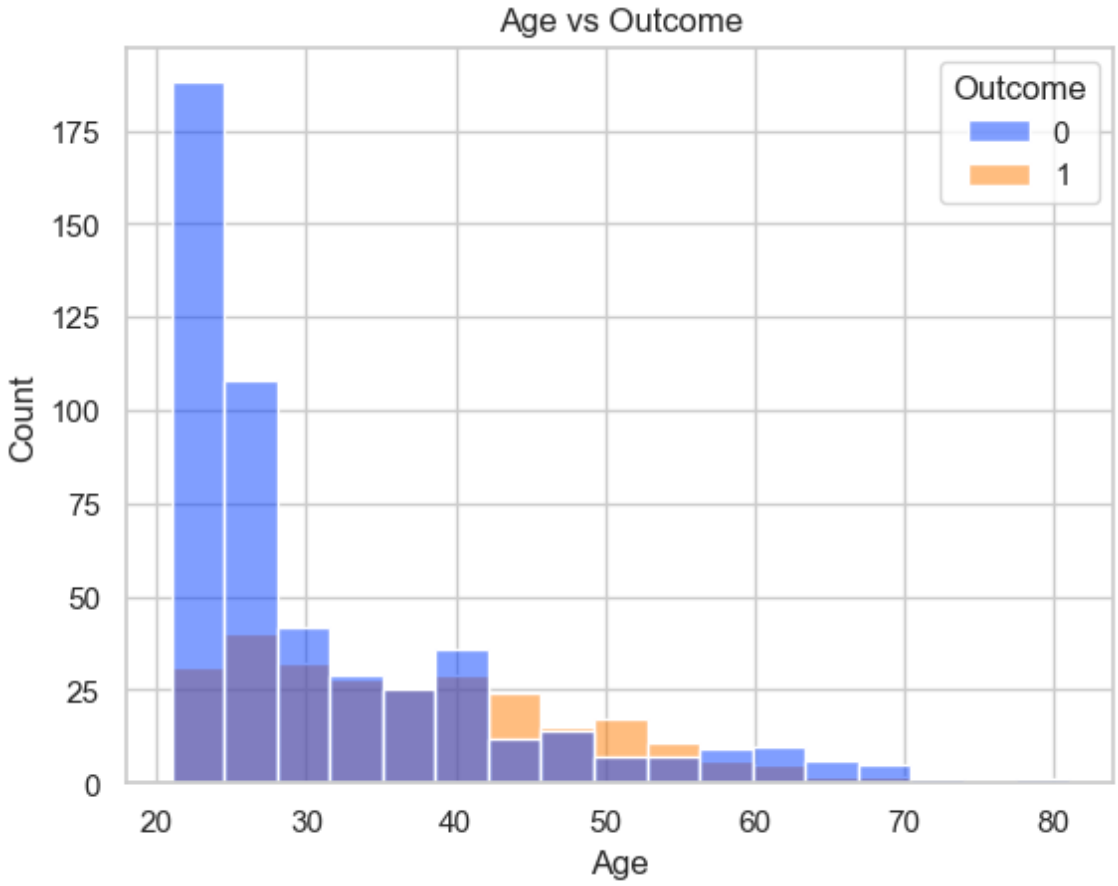
In [18]:

```
sns.scatterplot(x='BloodPressure',y='BMI',hue='Outcome',data=df)
plt.title('BloodPressure vs Outcome')
plt.show()
```



In [19]:

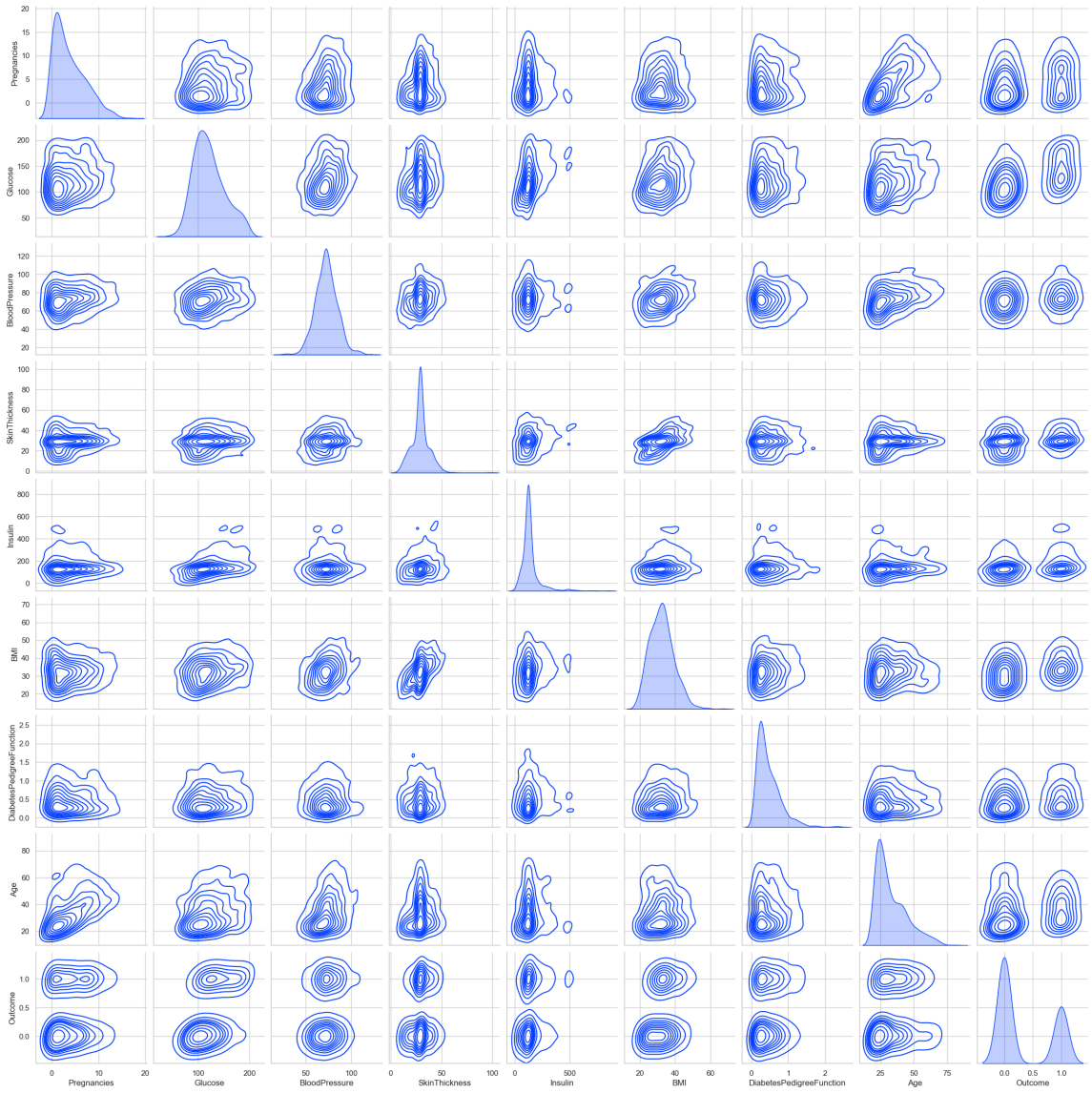
```
sns.histplot(x='Age',hue='Outcome', data=df)
plt.title('Age vs Outcome')
plt.show()
```



In [20]:

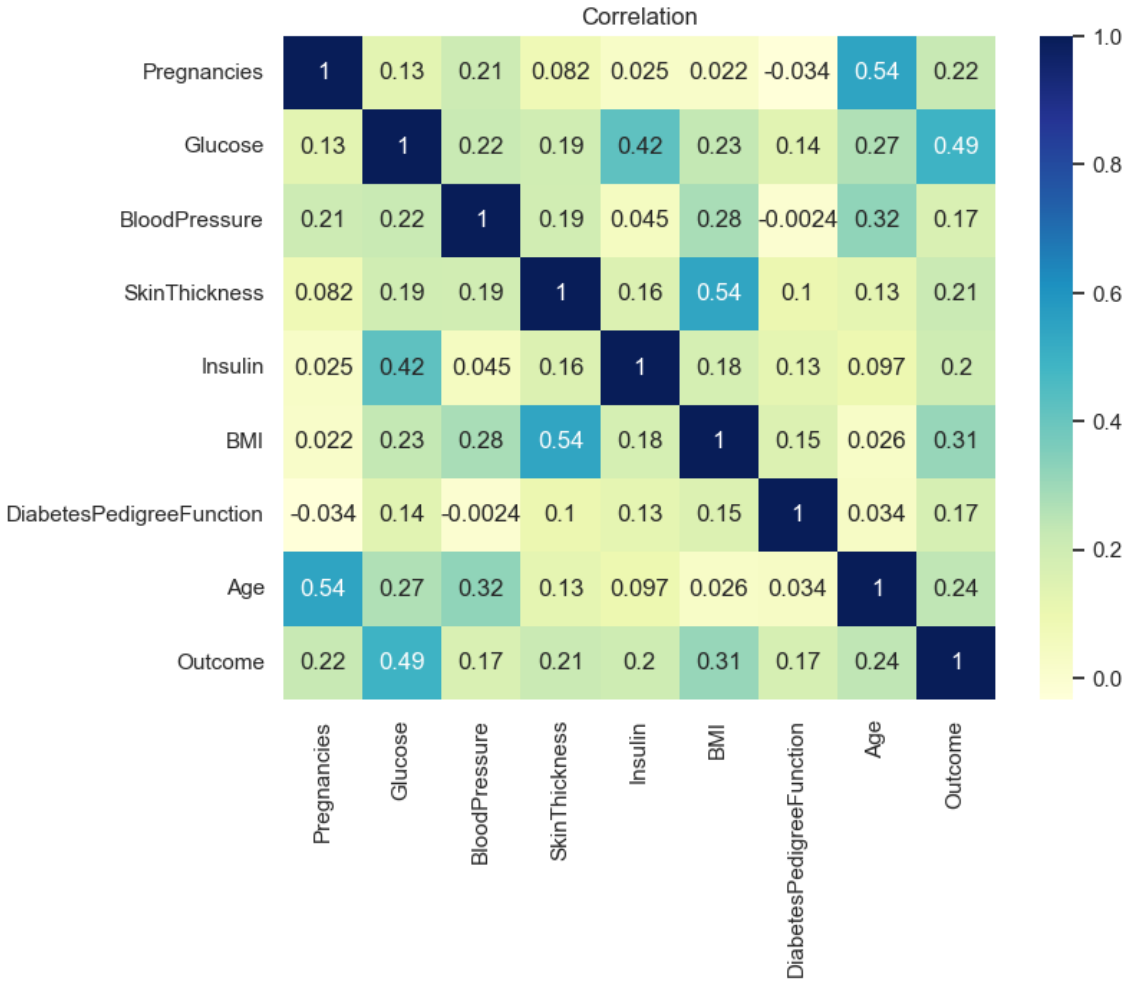
```
plt.figure(figsize=(25,20))
sns.pairplot(df, kind='kde')
plt.show()
```

```
<Figure size 2500x2000 with 0 Axes>
```



In [21]:

```
plt.figure(figsize=(8,6))
sns.heatmap(df.corr(),annot=True,cmap='YlGnBu')
plt.title('Correlation')
plt.show()
```



In [68]:

```
#BloodPressure & DiabetesPedigreeFunction are showing correlation
df.drop(['BloodPressure'],axis=1,inplace=True)  # dropping BloodPressure
```

Data Splitting

In [79]:

```
# split the data into independent and dependent variable
x = df.iloc[:,:-1]
y = df.iloc[:,-1]
```

In [80]:

```
x.head(2)
```

Out[80]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc
0	6	148.0	72.0	35.0	125.0	33.6	0.
1	1	85.0	66.0	29.0	125.0	26.6	0.

In [81]:

```
y.head(2)
```

Out[81]:

```
0    1
1    0
Name: Outcome, dtype: int64
```

In [82]:

```
# split the data into training and test
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
                                                    random_state=45)
```

Building Naive Bayes Classifier model

In [83]:

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x_train, y_train)
```

Out[83]:

```
GaussianNB
GaussianNB()
```

In [84]:

```
# Predict the model
y_pred_train = gnb.predict(x_train)
y_pred_test = gnb.predict(x_test)
```

In [85]:

```
# Evaluate
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
```

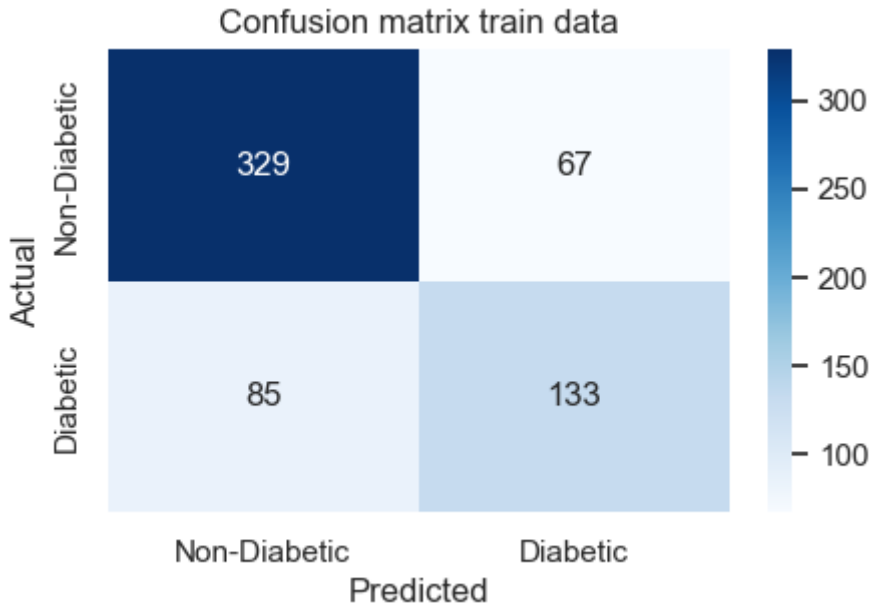
In [86]:

```
print(confusion_matrix(y_train,y_pred_train))
print(confusion_matrix(y_test,y_pred_test))
```

```
[[329  67]
 [ 85 133]]
[[86 18]
 [19 31]]
```

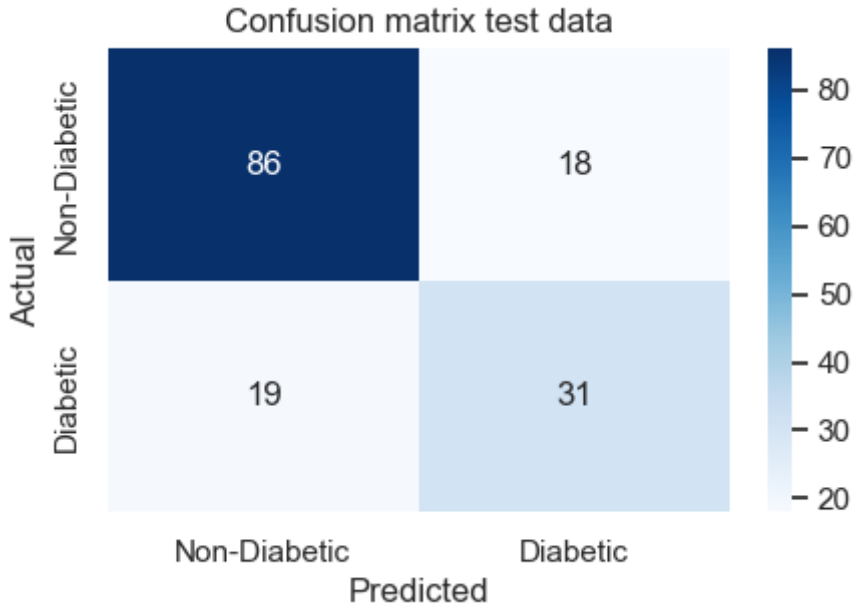
In [87]:

```
# plotting confusion matrix of train data
Labels = ['Non-Diabetic', 'Diabetic']
conf_matrix = confusion_matrix(y_train, y_pred_train)
plt.figure(figsize=(5,3))
sns.heatmap(conf_matrix,xticklabels= Labels, yticklabels=Labels,cmap='Blues',
            annot=True, fmt='g')
plt.title("Confusion matrix train data")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



In [88]:

```
#plotting confusion matrix of test data
Labels = ['Non-Diabetic', 'Diabetic']
conf_matrix = confusion_matrix(y_test, y_pred_test)
plt.figure(figsize=(5,3))
sns.heatmap(conf_matrix,xticklabels= Labels, yticklabels=Labels,cmap='Blues',
            annot=True, fmt='g')
plt.title("Confusion matrix test data")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



In [89]:

```
# Evaluate train data
acc= accuracy_score(y_train,y_pred_train)
print('Accuracy score of Train data is',acc)
print('---------------'*5)
acc= accuracy_score(y_test,y_pred_test)
print('Accuracy score of Test data is',acc)
```

```
Accuracy score of Train data is 0.752442996742671
--------------------------------------------------------------------------------
-
Accuracy score of Test data is 0.7597402597402597
```

In [91]:

```
# Cross validation
from sklearn.model_selection import cross_val_score
training_accuracy = cross_val_score(gnb,x_train, y_train, cv=10)
test_accuracy = cross_val_score(gnb,x_test, y_test, cv=10)
print('Accuracy of Train data after Cross validation:',
      training_accuracy.mean())
print('---------------'*5)
print('Accuracy of Test data after Cross validation:',
      test_accuracy.mean())
```

```
Accuracy of Train data after Cross validation: 0.7426493918561607
--------------------------------------------------------------------------------
-
Accuracy of Test data after Cross validation: 0.76
```

Conclusion

- I used Gaussian Naive Bayes algorithm to build the model.
- Train and test accuracy both are coming at 75%.
- After cross validation train accuracy is 74% and test accuracy is 76% making it a successful model to predict the results.

In [ ]:

```

```