In [33]:
```
import streamlit as st
import pandas as pd
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import plotly.figure_factory as ff
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import seaborn as sns
```

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

In [3]:
```
df.describe()
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	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	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

In [4]:
```
df.head()
```

Out[4]:

	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

In [5]:
```
df.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]:
```
st.title('Diabetes Checkup')
st.sidebar.header('Patient Data')
st.subheader('Training Data Stats')
```

In [8]:
```
x = df.drop(['Outcome'], axis = 1)
y = df.iloc[:, -1]
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.2, random_state = 0)
```

In [9]:
```
def user_report():
  pregnancies = st.sidebar.slider('Pregnancies', 0,17, 3 )
  glucose = st.sidebar.slider('Glucose', 0,200, 120 )
  bp = st.sidebar.slider('Blood Pressure', 0,122, 70 )
  skinthickness = st.sidebar.slider('Skin Thickness', 0,100, 20 )
  insulin = st.sidebar.slider('Insulin', 0,846, 79 )
  bmi = st.sidebar.slider('BMI', 0,67, 20 )
  dpf = st.sidebar.slider('Diabetes Pedigree Function', 0.0,2.4, 0.47 )
  age = st.sidebar.slider('Age', 21,88, 33 )
```

In [10]:
```
def user_report():
  pregnancies = st.sidebar.slider('Pregnancies', 0,17, 3 )
  glucose = st.sidebar.slider('Glucose', 0,200, 120 )
  bp = st.sidebar.slider('Blood Pressure', 0,122, 70 )
  skinthickness = st.sidebar.slider('Skin Thickness', 0,100, 20 )
  insulin = st.sidebar.slider('Insulin', 0,846, 79 )
  bmi = st.sidebar.slider('BMI', 0,67, 20 )
  dpf = st.sidebar.slider('Diabetes Pedigree Function', 0.0,2.4, 0.47 )
  age = st.sidebar.slider('Age', 21,88, 33 )

  user_report_data = {
      'pregnancies':pregnancies,
      'glucose':glucose,
      'bp':bp,
      'skinthickness':skinthickness,
      'insulin':insulin,
      'bmi':bmi,
      'dpf':dpf,
      'age':age
```

In [12]:
```
st.title('Visualised Patient Report')
```

Out[12]:
```
DeltaGenerator()
```

In [14]:
```
# Check if there is a function named 'model' defined in the current environment
if 'model' in globals():
    # Print a warning message
    print("Warning: There is a function named 'model' defined in the global environment.")
    print("This may cause unexpected behavior.")
```

In [15]:
```
st.header('Pregnancy count Graph (Others vs Yours)')
fig_preg = plt.figure()
ax1 = sns.scatterplot(x = 'Age', y = 'Pregnancies', data = df, hue = 'Outcome', palette = 'Greens')
ax2 = sns.scatterplot(x = user_data['Age'], y = user_data['pregnancies'], s = 150, color = 'orange')
plt.xticks(np.arange(10,100,5))
plt.yticks(np.arange(0,20,2))
plt.title('0 - Healthy & 1 - Unhealthy')
st.pyplot(fig_preg)
```

Out[15]:
```
DeltaGenerator()
```

In [16]:
```
# Check if the 'glucose' column exists in the user_data DataFrame
if 'glucose' not in user_data.columns:
    # Add the 'glucose' column to the user_data DataFrame
    user_data['glucose'] = 120  # Replace with the appropriate value

# Check if the 'glucose' column is not empty
if user_data['glucose'].isnull().any():
    # Fill missing values in the 'glucose' column with an appropriate value
    user_data['glucose'].fillna(120, inplace=True)  # Replace with the appropriate value

# Now you can use the 'glucose' column to create the scatterplot
ax4 = sns.scatterplot(x = user_data['Age'], y = user_data['glucose'], s = 150, color = 'green')
```

In [17]:
```
st.header('Blood Pressure Value Graph (Others vs Yours)')
```

In [18]:
```
st.header('Skin Thickness Value Graph (Others vs Yours)')
fig_st = plt.figure()
ax7 = sns.scatterplot(x = 'Age', y = 'SkinThickness', data = df, hue = 'Outcome', palette='Blues')
ax8 = sns.scatterplot(x = user_data['Age'], y = user_data['skinthickness'], s = 150, color = 'blue')
plt.xticks(np.arange(10,100,5))
plt.yticks(np.arange(0,110,10))
plt.title('0 - Healthy & 1 - Unhealthy')
st.pyplot(fig_st)
```

Out[18]:
```
DeltaGenerator()
```

In [19]:
```
st.header('Insulin Value Graph (Others vs Yours)')
fig_i = plt.figure()
ax9 = sns.scatterplot(x = 'Age', y = 'Insulin', data = df, hue = 'Outcome', palette='rocket')
ax10 = sns.scatterplot(x = user_data['Age'], y = user_data['insulin'], s = 150, color = 'blue')
plt.xticks(np.arange(10,100,5))
plt.yticks(np.arange(0,900,50))
plt.title('0 - Healthy & 1 - Unhealthy')
st.pyplot(fig_i)
```

Out[19]:
```
DeltaGenerator()
```

In [20]:
```
if 'bmi' not in user_data.columns:
    # Add the 'bmi' column to the user_data DataFrame
    user_data['bmi'] = 25  # Replace with the appropriate value

# Now you can use the 'bmi' column to create the scatterplot
ax12 = sns.scatterplot(x = user_data['Age'], y = user_data['bmi'], s = 150, color = 'green')
```

In [21]:
```
try:
    # Try to create the scatterplot using the 'bmi' column
    ax12 = sns.scatterplot(x = user_data['Age'], y = user_data['bmi'], s = 150, color = 'green')
except KeyError:
```

In [24]:
```
if 'dpf' in user_data.columns:
    # Create the scatterplot using the 'dpf' column
    ax14 = sns.scatterplot(x = user_data['Age'], y = user_data['dpf'], s = 150, color = 'blue')
else:
    # Print a message indicating that the 'dpf' column does not exist
    print("The 'dpf' column does not exist in the user_data DataFrame.")
```

```
The 'dpf' column does not exist in the user_data DataFrame.
```

In [29]:
```
# Import necessary libraries
import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Load the dataset
df = pd.read_csv('diabetes.csv')

# Define the user's data
user_data = {
    'Pregnancies': [3],
    'Glucose': [130],
    'BloodPressure': [78],
    'SkinThickness': [0],
    'Insulin': [0],
    'BMI': [33.6],
    'DiabetesPedigreeFunction': [0.627],
    'Age': [47]
}

# Convert user_data to a DataFrame
user_data_df = pd.DataFrame(user_data)
```

```
rf = RandomForestClassifier()
rf.fit(x, y)

# Make predictions for the user's data
user_result = rf.predict(user_data_df)

# Output the prediction result
st.write("Prediction for the user's data: ", "Diabetic" if user_result[0] == 1 else "Not Diabetic")

# Make predictions on the entire dataset
y_pred = rf.predict(x)

# Calculate accuracy, precision, and recall
accuracy = accuracy_score(y, y_pred)
precision = precision_score(y, y_pred)
recall = recall_score(y, y_pred)

# Display the performance metrics
st.write(f"Model Accuracy: {accuracy:.2f}")
st.write(f"Model Precision: {precision:.2f}")
st.write(f"Model Recall: {recall:.2f}")

# Create the BMI scatterplot
fig_bmi = plt.figure()
ax = sns.scatterplot(data=df, x='BMI', y='Age', hue='Outcome')
plt.title('BMI vs Age Scatterplot')
plt.xlabel('BMI')
plt.ylabel('Age')

# Display the scatterplot in Streamlit
st.pyplot(fig_bmi)
```

Out[32]:
```
DeltaGenerator()
```