Exploratory Data Analysis and Prediction on Diabetes Data

1. Load libraries and read the data

1.1. Load libraries

In [5]:
```
# Classic,data manipulation and linear algebra
import pandas as pd
```

```
import numpy as np

# Plots
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import plotly.offline as py
import plotly.graph_objs as go
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
import plotly.tools as tls
import plotly.figure_factory as ff
py.init_notebook_mode(connected=True)
import squarify

# Data processing, metrics and modeling
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import GridSearchCV, cross_val_score, train_test_split, GridSearchCV, RandomizedSearchCV
from sklearn.metrics import precision_score, recall_score, confusion_matrix,  roc_curve, precision_recall_curve, accuracy_score, roc_auc_score
import lightgbm as lgbm
from sklearn.ensemble import VotingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve,auc
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_predict
from yellowbrick.classifier import DiscriminationThreshold

# Stats
import scipy.stats as ss
from scipy import interp
from scipy.stats import randint as sp_randint
from scipy.stats import uniform as sp_uniform

# Time
from contextlib import contextmanager
@contextmanager
def timer(title):
    t0 = time.time()
    yield
    print("{} - done in {:.0f}s".format(title, time.time() - t0))

#ignore warning messages
import warnings
warnings.filterwarnings('ignore')
```

1.2. Read data

In [6]:
```
data = pd.read_csv('C:/Users/diabetes.csv')
```

2. Overview

2.1. Head

Checking data head and info

In [7]:
```
display(data.info(),data.head())
```

```
<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
None
```

	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

The datasets consist of several medical predictor (independent) variables and one target (dependent) variable, 'Outcome.' Independent variables include the patient's number of pregnancies, BMI, insulin level, age, and more.

What is Diabetes ?

Diabetes is a disease that occurs when blood glucose, also known as blood sugar, is too high. Blood glucose serves as the primary source of energy derived from the food we eat. Insulin, a hormone produced by the pancreas, assists in moving glucose from food into cells for energy. When the body doesn’t produce enough insulin or can’t effectively use it, glucose remains in the blood and fails to reach the cells.

Persistently high blood glucose levels over time can lead to various health problems. While diabetes currently has no cure, managing the condition through appropriate measures can help maintain overall health.

Sometimes, diabetes is referred to as 'a touch of sugar' or 'borderline diabetes.' These terms might imply a less serious case or the absence of diabetes, but it's important to recognize that every case of diabetes is significant.

Different types of diabetes include the most common forms: type 1, type 2, and gestational diabetes.

Type 1 diabetes: In this type, the body doesn’t produce insulin. Typically diagnosed in children and young adults, the immune system attacks and destroys the insulin-producing cells in the pancreas. Individuals with type 1 diabetes require daily insulin injections to sustain life.

Type 2 diabetes: In this type, the body either doesn't produce enough insulin or doesn't use it efficiently. Type 2 diabetes can develop at any age, even during childhood, but it most commonly occurs in middle-aged and older people. It's the most prevalent form of diabetes.

Gestational diabetes: This type develops in some women during pregnancy. Often, it resolves after childbirth. However, having had gestational diabetes increases the risk of developing type 2 diabetes later in life. In some cases, diabetes diagnosed during pregnancy may actually be type 2 diabetes.

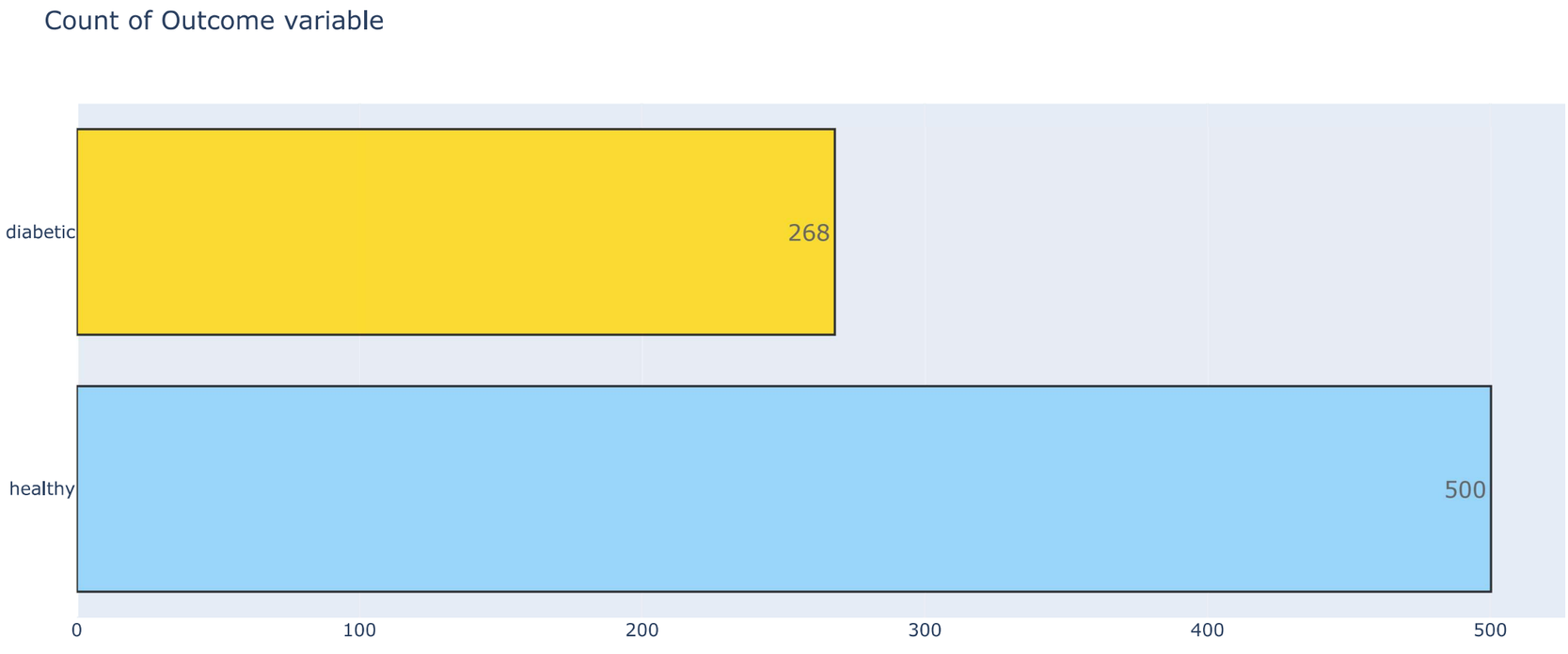
Other types of diabetes: Less common forms include monogenic diabetes, an inherited type, and cystic fibrosis-related diabetes.

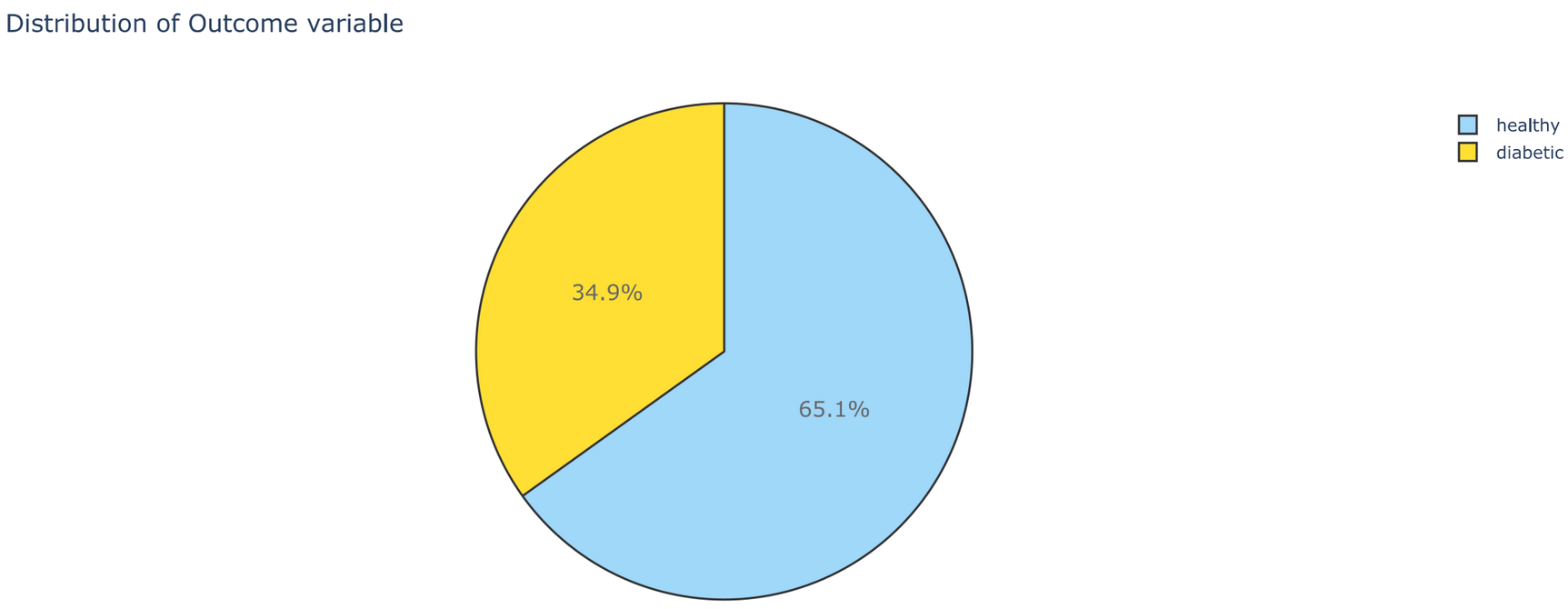
2.2 Target

The target's distribution.

In [8]:
```
# 2 datasets
D = data[(data['Outcome'] != 0)]
H = data[(data['Outcome'] == 0)]

#------------COUNT-----------------------
def target_count():
    trace = go.Bar( x = data['Outcome'].value_counts().values.tolist(),
                    y = ['healthy','diabetic' ],
                    orientation = 'h',
                    text=data['Outcome'].value_counts().values.tolist(),
                    textfont=dict(size=15),
                    textposition = 'auto',
                    opacity = 0.8,marker=dict(
                    color=['lightskyblue', 'gold'],
                    line=dict(color='#000000',width=1.5)))

    layout = dict(title =  'Count of Outcome variable')

    fig = dict(data = [trace], layout=layout)
    py.iplot(fig)

#------------PERCENTAGE-------------------
def target_percent():
    trace = go.Pie(labels = ['healthy','diabetic'], values = data['Outcome'].value_counts(),
                   textfont=dict(size=15), opacity = 0.8,
                   marker=dict(colors=['lightskyblue', 'gold'],
                               line=dict(color='#000000', width=1.5)))


    layout = dict(title =  'Distribution of Outcome variable')

    fig = dict(data = [trace], layout=layout)
    py.iplot(fig)
```

In [108…
```
target_count()
target_percent()
```


Count of Outcome variable
diabetic
268
healthy
500
0
100
200
300
400
500



The above graph shows that the data is unbalanced. The number of non-diabetic is 268, the number of diabetic patients is 500.

2.3. Missing values

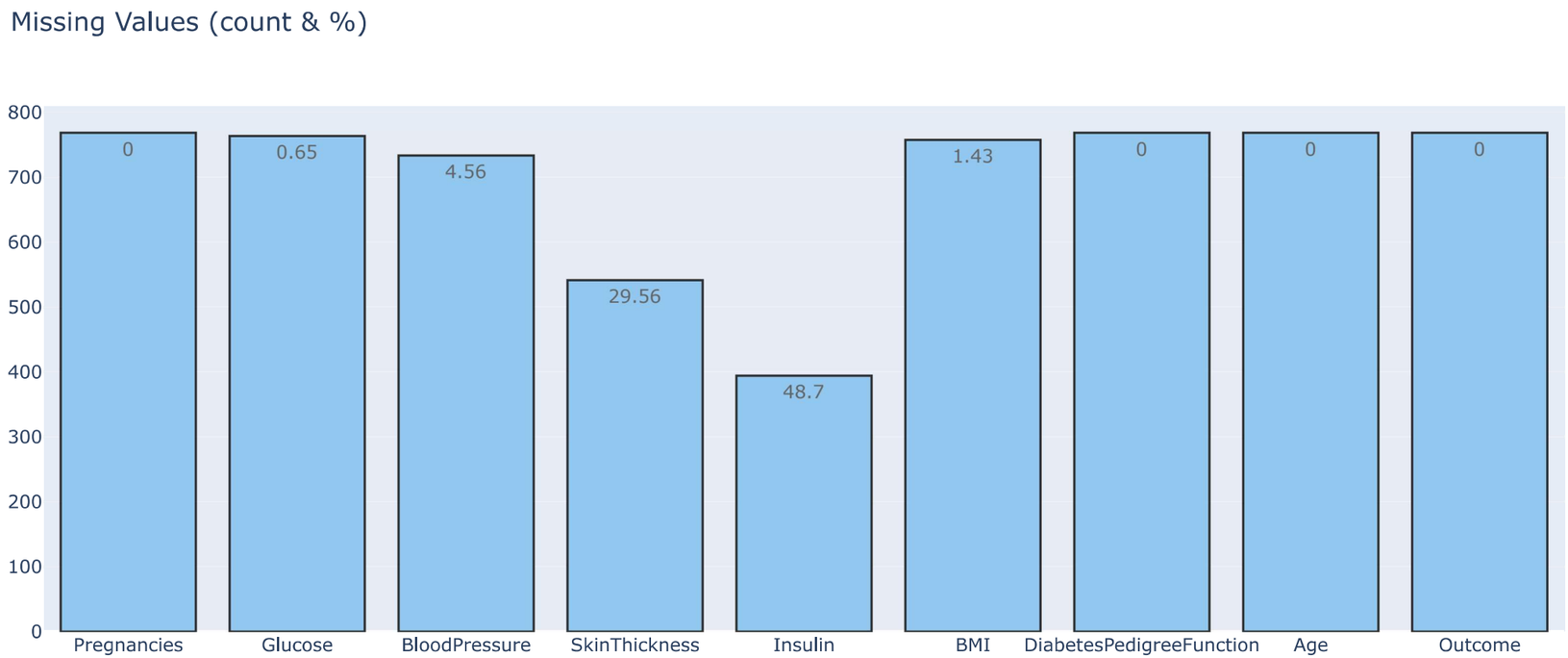
We observed in data.head() that certain features contain zeros. This seems inappropriate and indicates missing values. As a solution, we replace these 0 values with NaN.

In [10]:
```
data[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = data[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']].replace(0,np.NaN)
```

Now, we can look at where are missing values :

In [11]:
```
# Define missing plot to detect all missing values in dataset
def missing_plot(dataset, key) :
    null_feat = pd.DataFrame(len(dataset[key]) - dataset.isnull().sum(), columns = ['Count'])
    percentage_null = pd.DataFrame((len(dataset[key]) - (len(dataset[key]) - dataset.isnull().sum()))/len(dataset[key])*100, columns = ['Count'])
    percentage_null = percentage_null.round(2)

    trace = go.Bar(x = null_feat.index, y = null_feat['Count'] ,opacity = 0.8, text = percentage_null['Count'],  textposition = 'auto',marker=dict(color = '#7EC0EE',
            line=dict(color='#000000',width=1.5)))

    layout = dict(title =  "Missing Values (count & %)")

    fig = dict(data = [trace], layout=layout)
    py.iplot(fig)
```

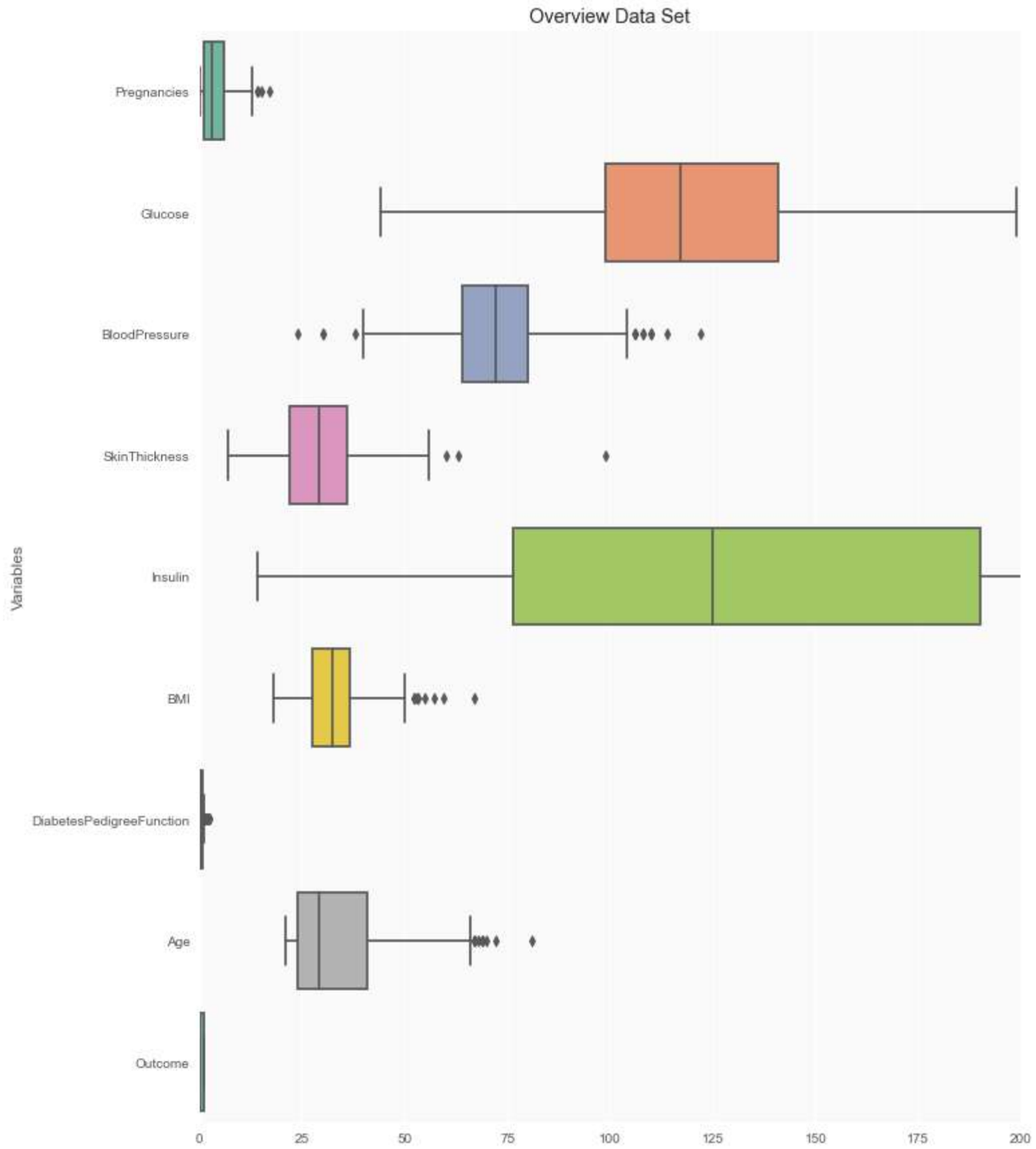
In [12]:
```
# Plotting
missing_plot(data, 'Outcome')
```



Missing values :

- Insulin = 48.7%
- SkinThickness = 29.56%
- BloodPressure = 4.56%
- BMI = 1.43%
- Glucose = 0.65%

In [13]:
```
plt.style.use('ggplot') # Using ggplot2 style visuals

f, ax = plt.subplots(figsize=(11, 15))

ax.set_facecolor('#fafafa')
ax.set(xlim=(-.05, 200))
plt.ylabel('Variables')
plt.title("Overview Data Set")
ax = sns.boxplot(data = data,
  orient = 'h',
  palette = 'Set2')
```



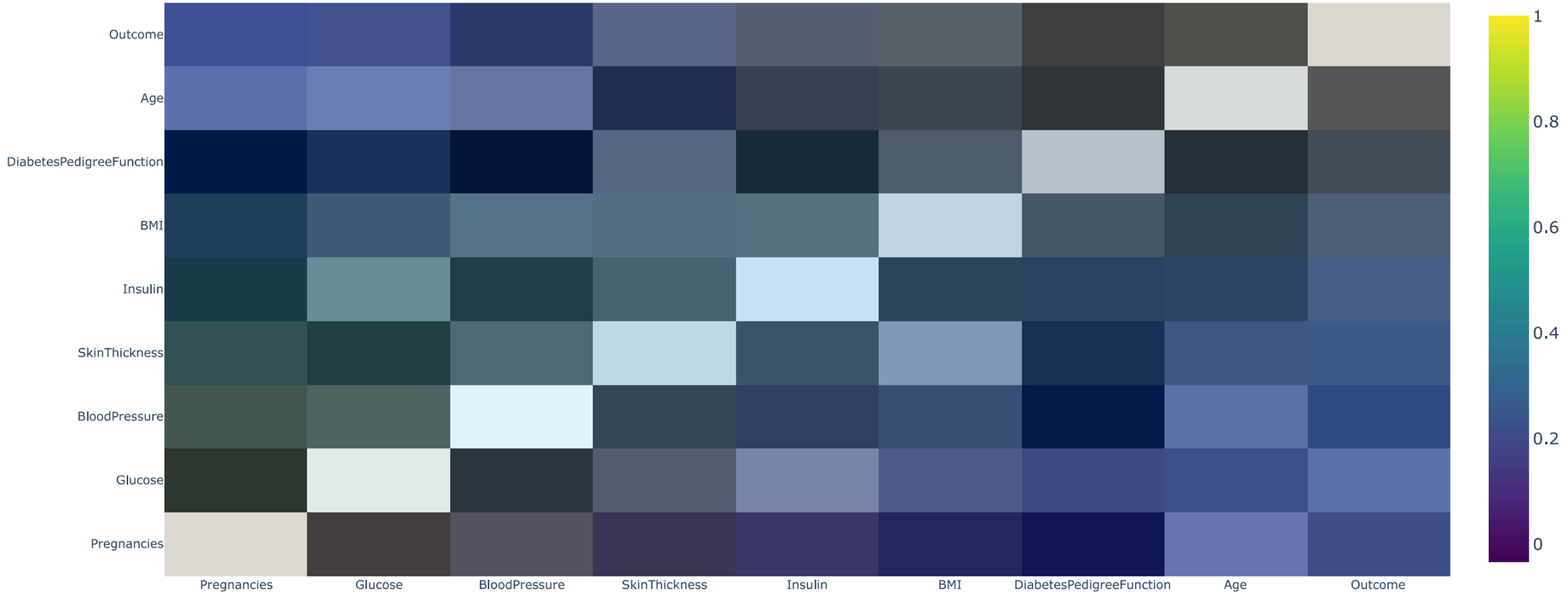
To properly fill these NaN values, it's necessary to comprehend the data distribution in relation to the target.

In [14]:
```
def correlation_plot():
    #correlation
    correlation = data.corr()
```

```
    #tick labels
    matrix_cols = correlation.columns.tolist()
    #convert to array
    corr_array  = np.array(correlation)
    trace = go.Heatmap(z = corr_array,
                       x = matrix_cols,
                       y = matrix_cols,
                       colorscale='Viridis',
                       colorbar   = dict() ,
                      )
    layout = go.Layout(dict(title = 'Correlation Matrix for variables',
                            #autosize = False,
                            #height  = 1400,
                            #width   = 1600,
                            margin  = dict(r = 0 ,l = 100,
                                           t = 0,b = 100,
                                         ),
                            yaxis   = dict(tickfont = dict(size = 9)),
                            xaxis   = dict(tickfont = dict(size = 9)),
                           )
                      )
    fig = go.Figure(data = [trace],layout = layout)
    py.iplot(fig)
```

A correlation matrix is a table that displays correlation coefficients between sets of variables. Each random variable (Xi) in the table is correlated with every other value (Xj) present. This visualization enables the identification of pairs with the highest correlation.

In [15]:
```
correlation_plot()
```



3. Replace missing values and EDA

In [16]:
```
def median_target(var):
    temp = data[data[var].notnull()]
    temp = temp[[var, 'Outcome']].groupby(['Outcome'])[[var]].median().reset_index()
    return temp
```

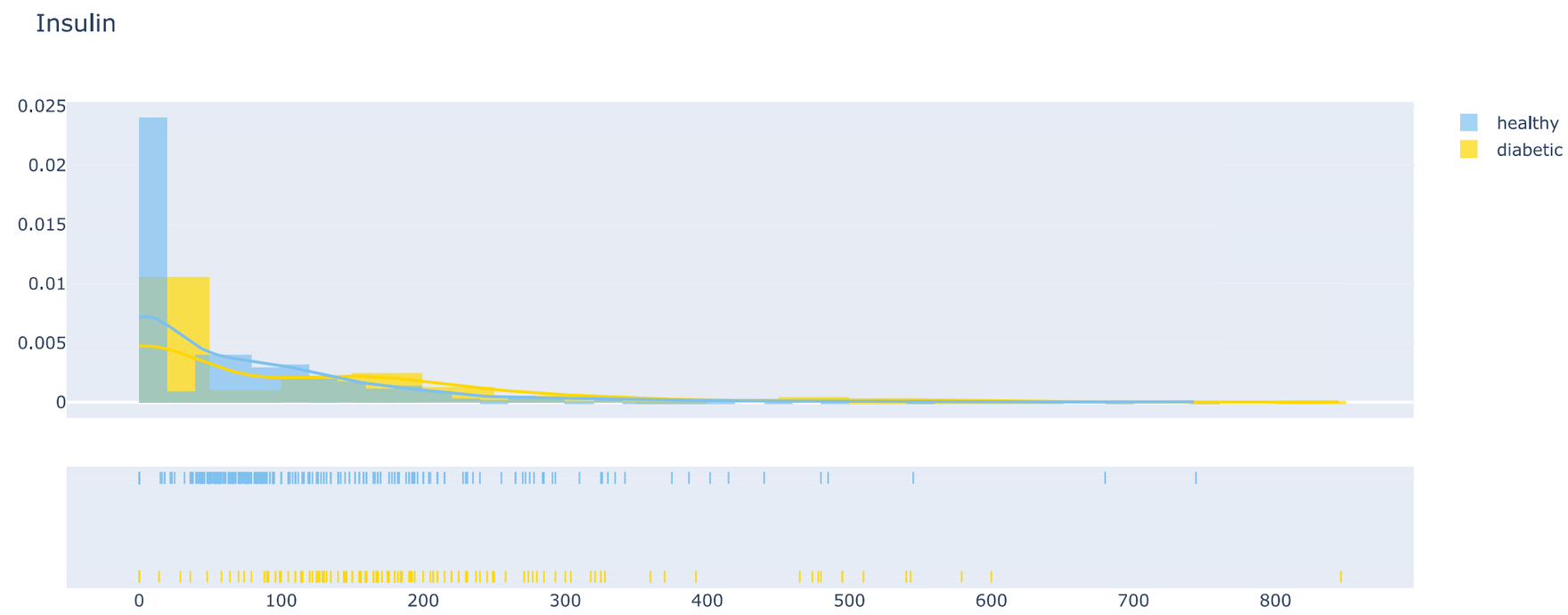
3.1. Insulin

- **Insulin** : 2-Hour serum insulin (mu U/ml)

In [17]:
```
def plot_distribution(data_select, size_bin) :
    # 2 datasets
    tmp1 = D[data_select]
    tmp2 = H[data_select]
    hist_data = [tmp1, tmp2]

    group_labels = ['diabetic', 'healthy']
```

```
    colors = ['#FFD700', '#7EC0EE']

    fig = ff.create_distplot(hist_data, group_labels, colors = colors, show_hist = True, bin_size = size_bin, curve_type='kde')

    fig['layout'].update(title = data_select)

    py.iplot(fig, filename = 'Density plot')
```

In [18]:
```
plot_distribution('Insulin', 0)
```



In [19]:
```
median_target('Insulin')
```

Out[19]:

	Outcome	Insulin
0	0	102.5
1	1	169.5

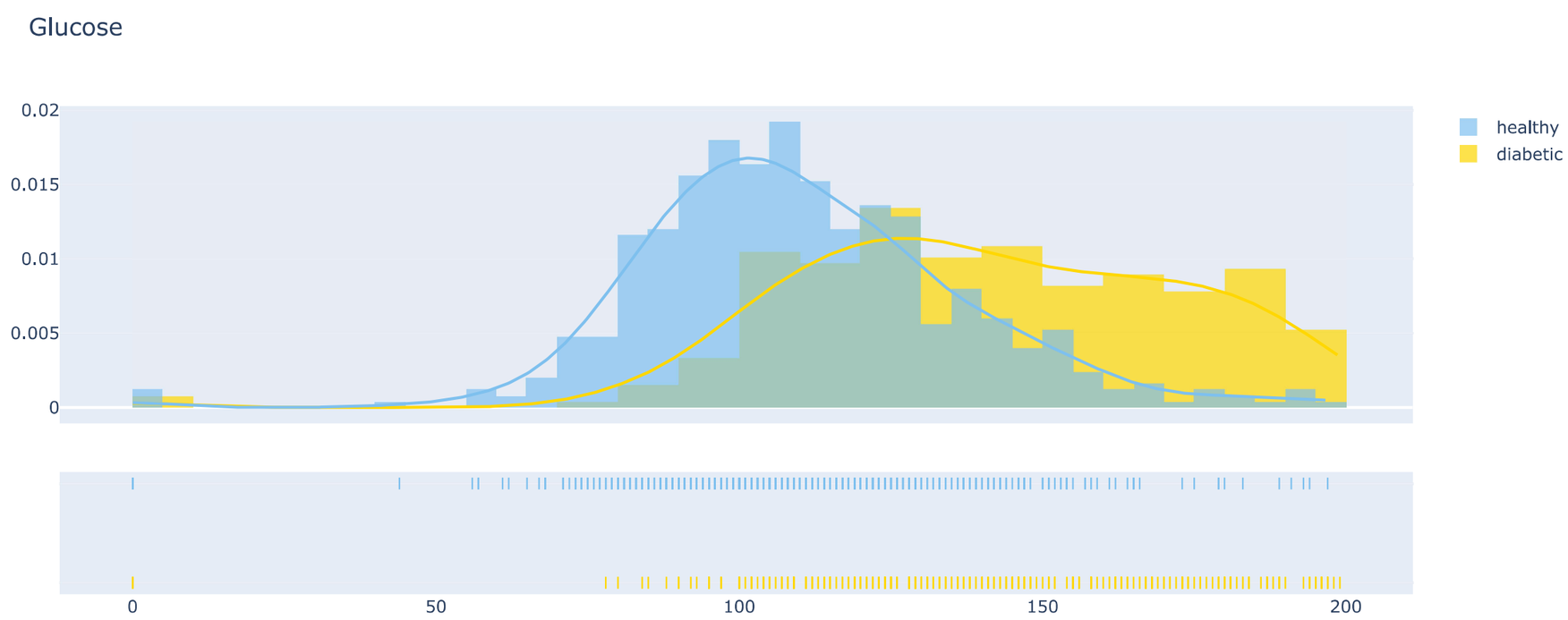
Insulin's medians by the target are really different. 102.5 for a healthy person and 169.5 for a diabetic person.

In [20]:
```
data.loc[(data['Outcome'] == 0 ) & (data['Insulin'].isnull()), 'Insulin'] = 102.5
data.loc[(data['Outcome'] == 1 ) & (data['Insulin'].isnull()), 'Insulin'] = 169.5
```

3.2. Glucose

- **Glucose** : Plasma glucose concentration a 2 hours in an oral glucose tolerance test

In [21]:
```
plot_distribution('Glucose', 0)
```



In [22]:
```
median_target('Glucose')
```

Out[22]:

	Outcome	Glucose
0	0	107.0
1	1	140.0

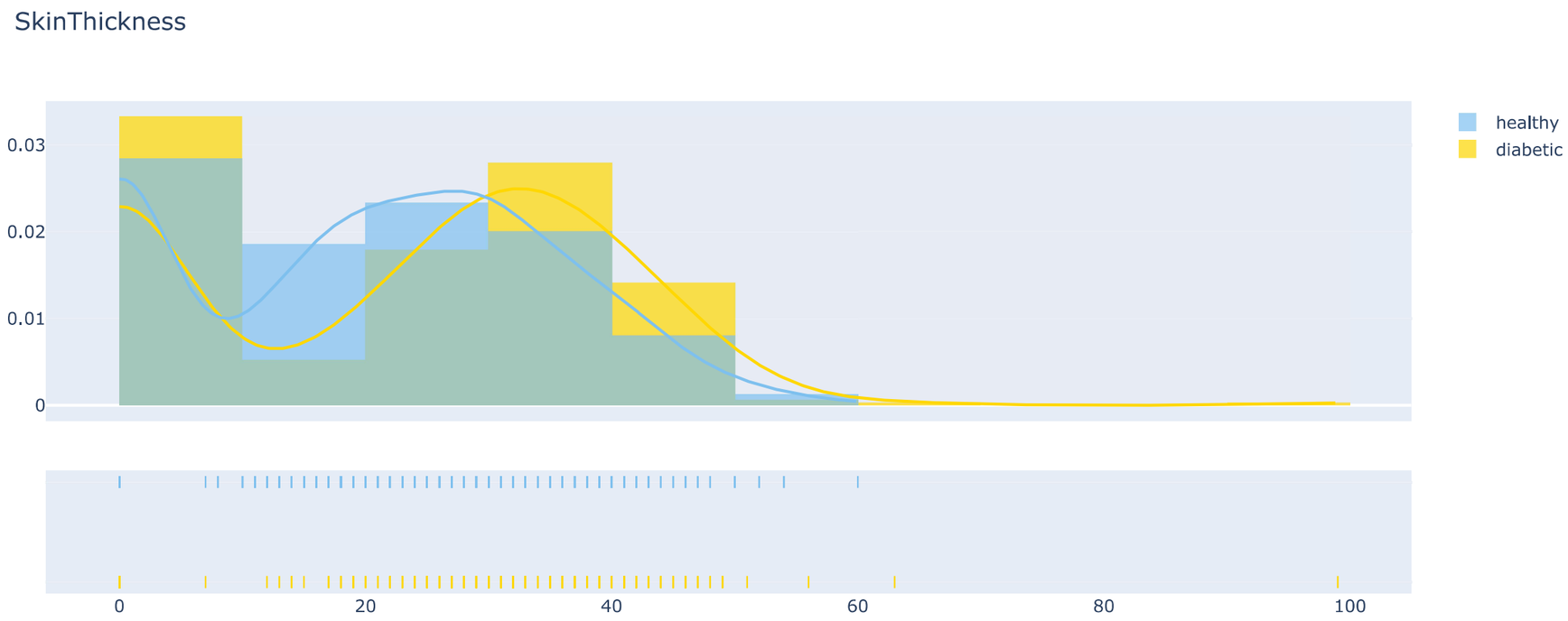
In [23]:
```
data.loc[(data['Outcome'] == 0 ) & (data['Glucose'].isnull()), 'Glucose'] = 107
data.loc[(data['Outcome'] == 1 ) & (data['Glucose'].isnull()), 'Glucose'] = 140
```

107 for a healthy person and 140 for a diabetic person.

3.3. SkinThickness

- **SkinThickness** : Triceps skin fold thickness (mm)

In [24]:
```
plot_distribution('SkinThickness', 10)
```



In [25]:
```
median_target('SkinThickness')
```

Out[25]:

	Outcome	SkinThickness
0	0	27.0
1	1	32.0

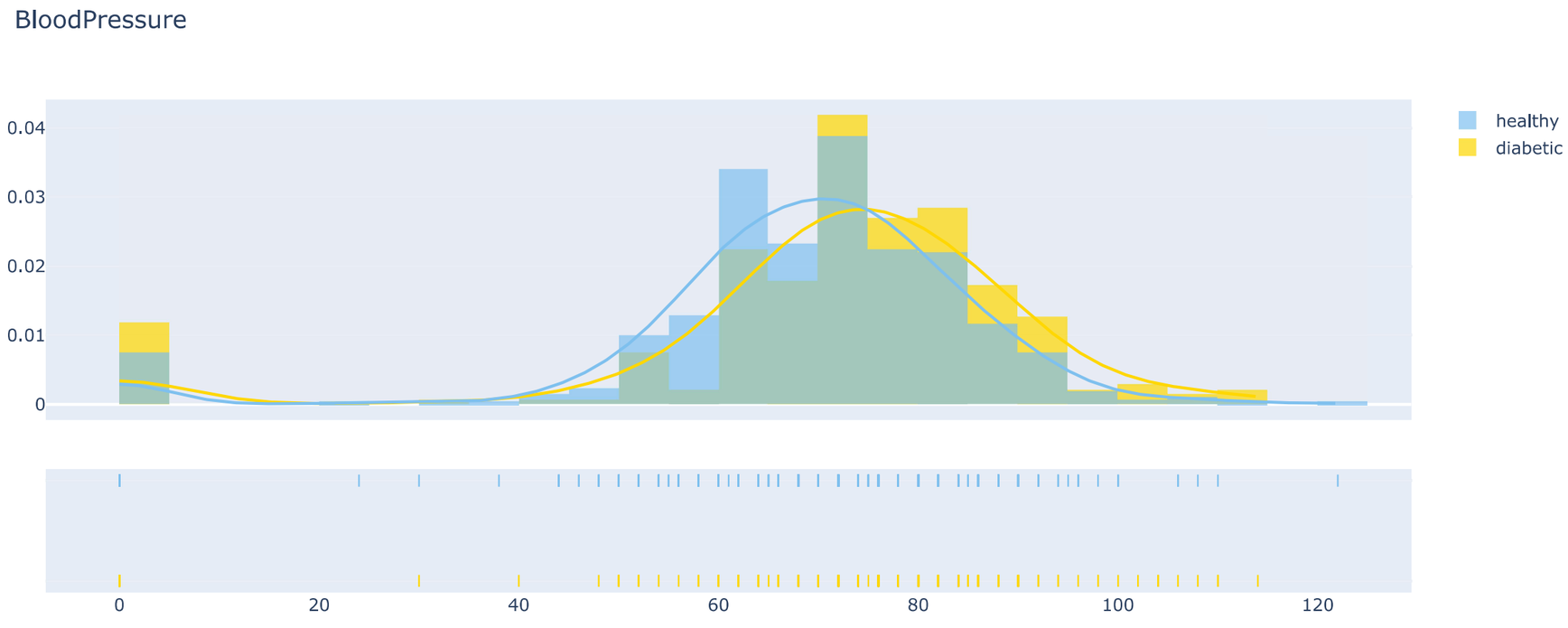
In [26]:
```
data.loc[(data['Outcome'] == 0 ) & (data['SkinThickness'].isnull()), 'SkinThickness'] = 27
data.loc[(data['Outcome'] == 1 ) & (data['SkinThickness'].isnull()), 'SkinThickness'] = 32
```

27 for a healthy person and 32 for a diabetic person.

3.4. BloodPressure

- **BloodPressure** : Diastolic blood pressure (mm Hg)

In [27]:
```
plot_distribution('BloodPressure', 5)
```



In [28]:
```
median_target('BloodPressure')
```

Out[28]:

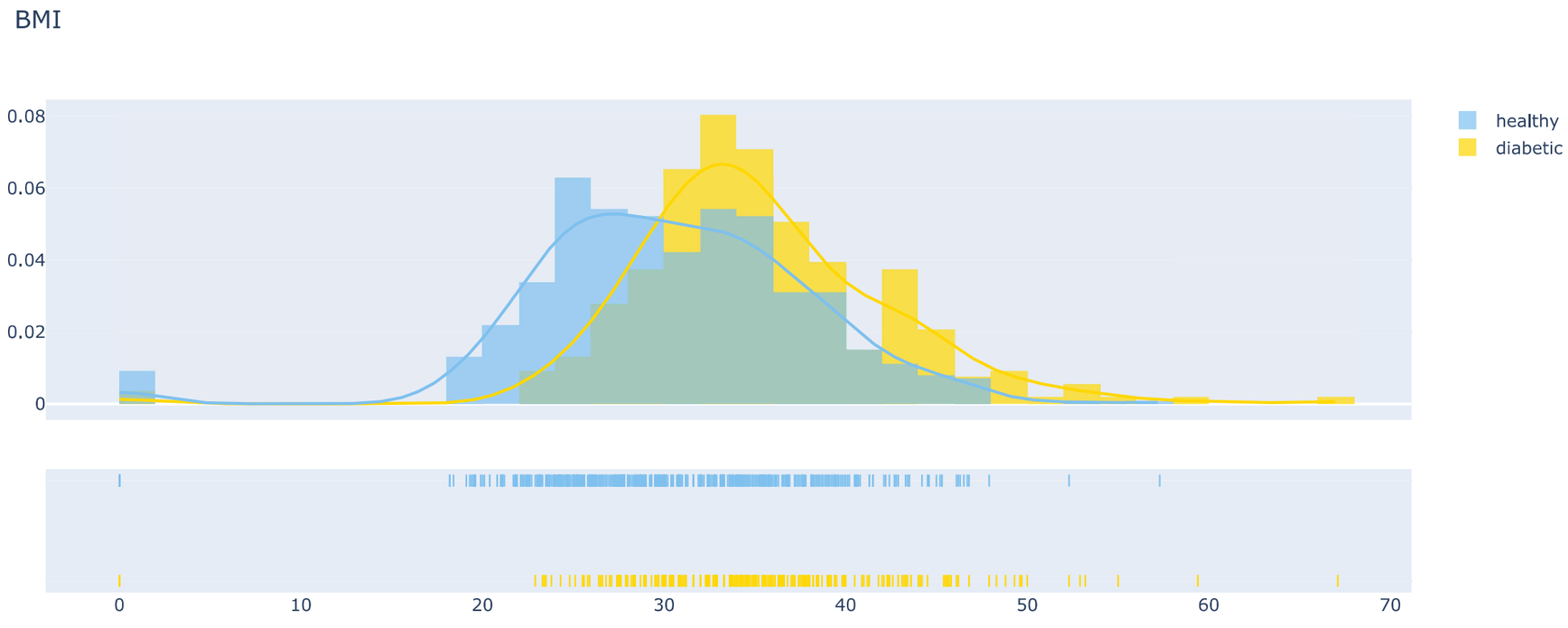
	Outcome	BloodPressure
0	0	70.0
1	1	74.5

In [29]:
```
data.loc[(data['Outcome'] == 0 ) & (data['BloodPressure'].isnull()), 'BloodPressure'] = 70
data.loc[(data['Outcome'] == 1 ) & (data['BloodPressure'].isnull()), 'BloodPressure'] = 74.5
```

3.5. BMI

- **BMI** : Body mass index (weight in kg/(height in m)^2)

In [30]:
```
plot_distribution('BMI', 0)
```



In [31]:
```
median_target('BMI')
```

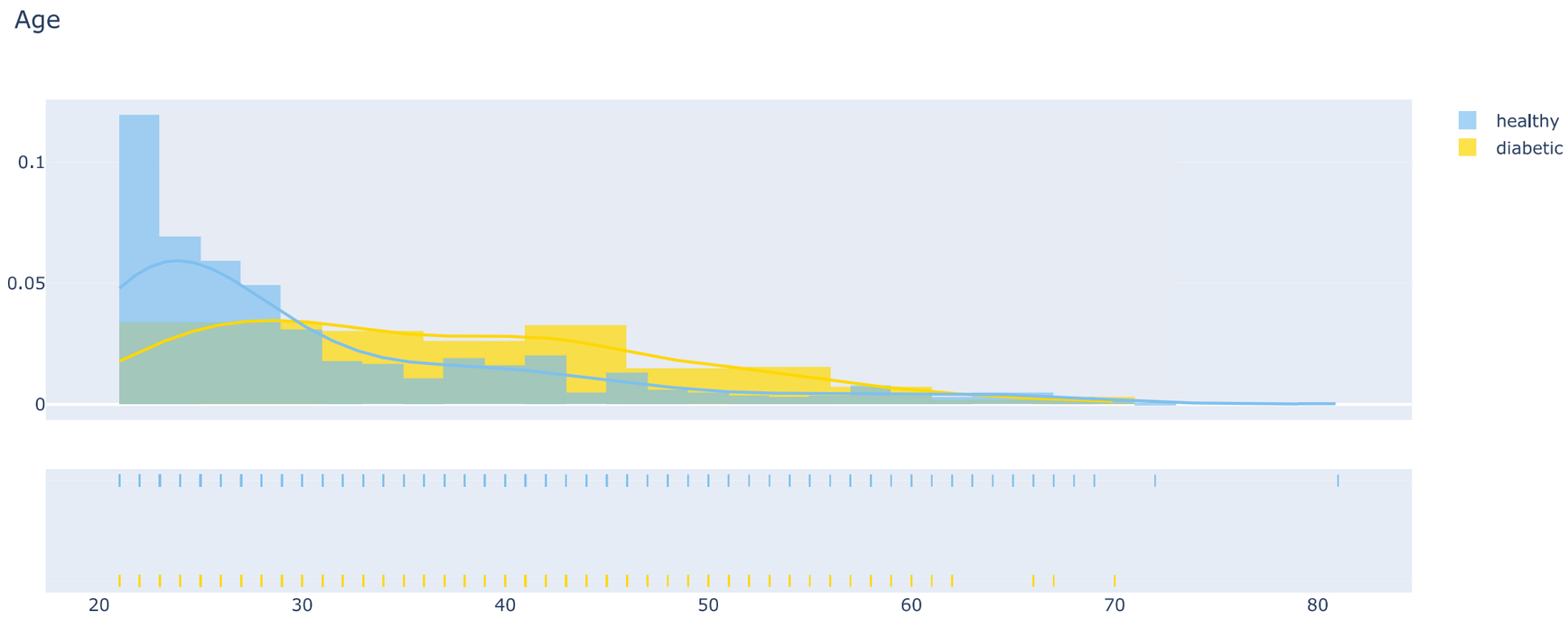
Out[31]:

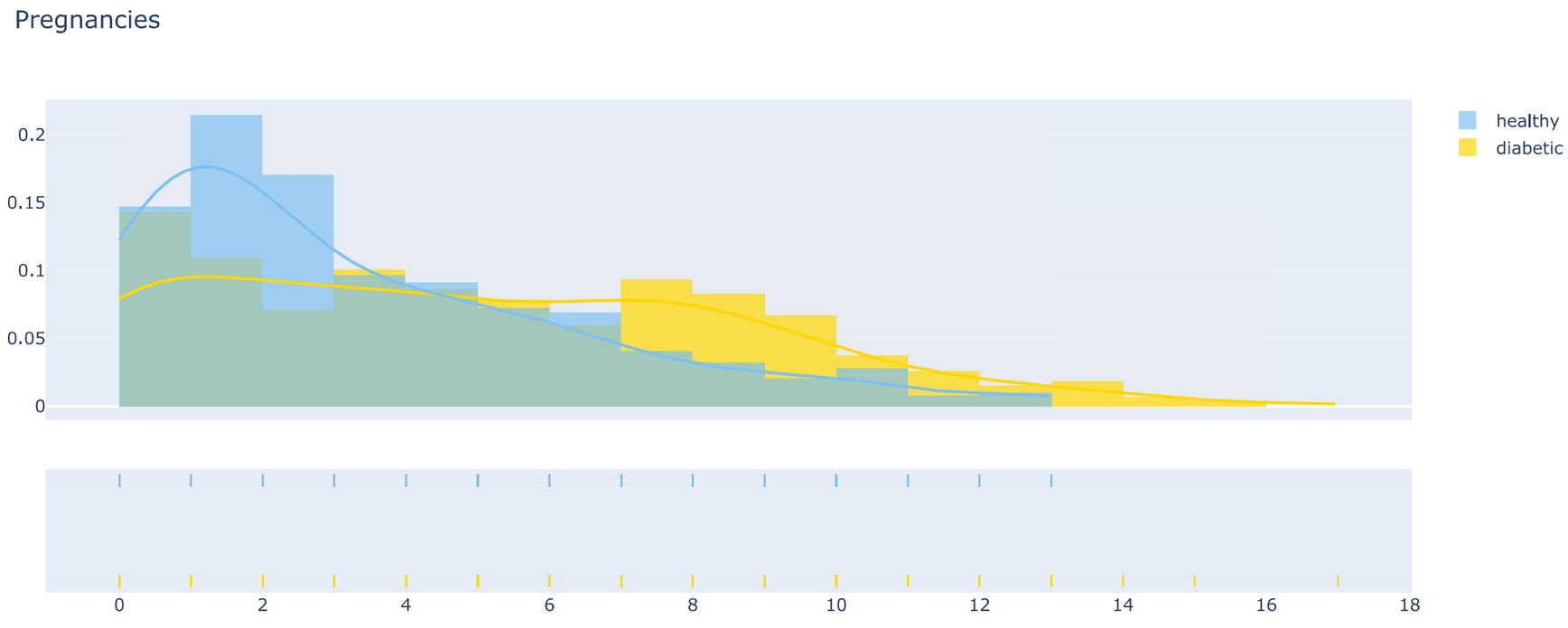
	Outcome	BMI
0	0	30.1
1	1	34.3

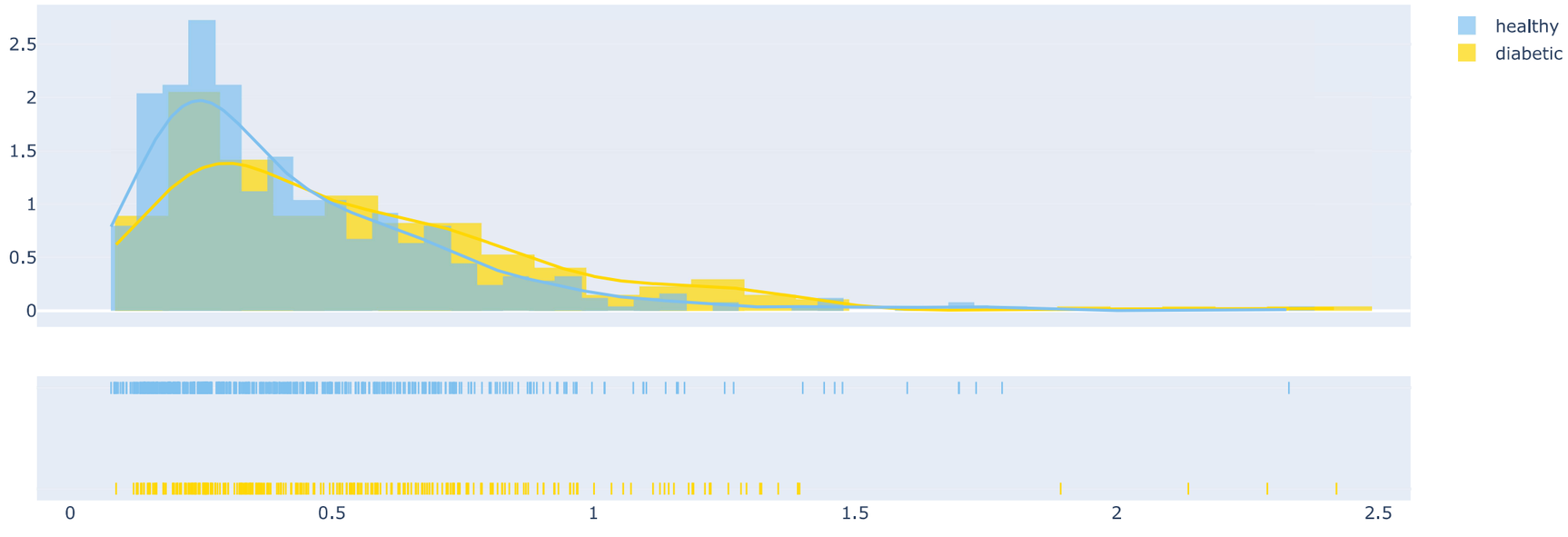
In [32]:
```
data.loc[(data['Outcome'] == 0 ) & (data['BMI'].isnull()), 'BMI'] = 30.1
data.loc[(data['Outcome'] == 1 ) & (data['BMI'].isnull()), 'BMI'] = 34.3
```

- **Age** : Age (years)
- **DiabetesPedigreeFunction** : Diabetes pedigree function
- **Pregnancies** : Number of times pregnant

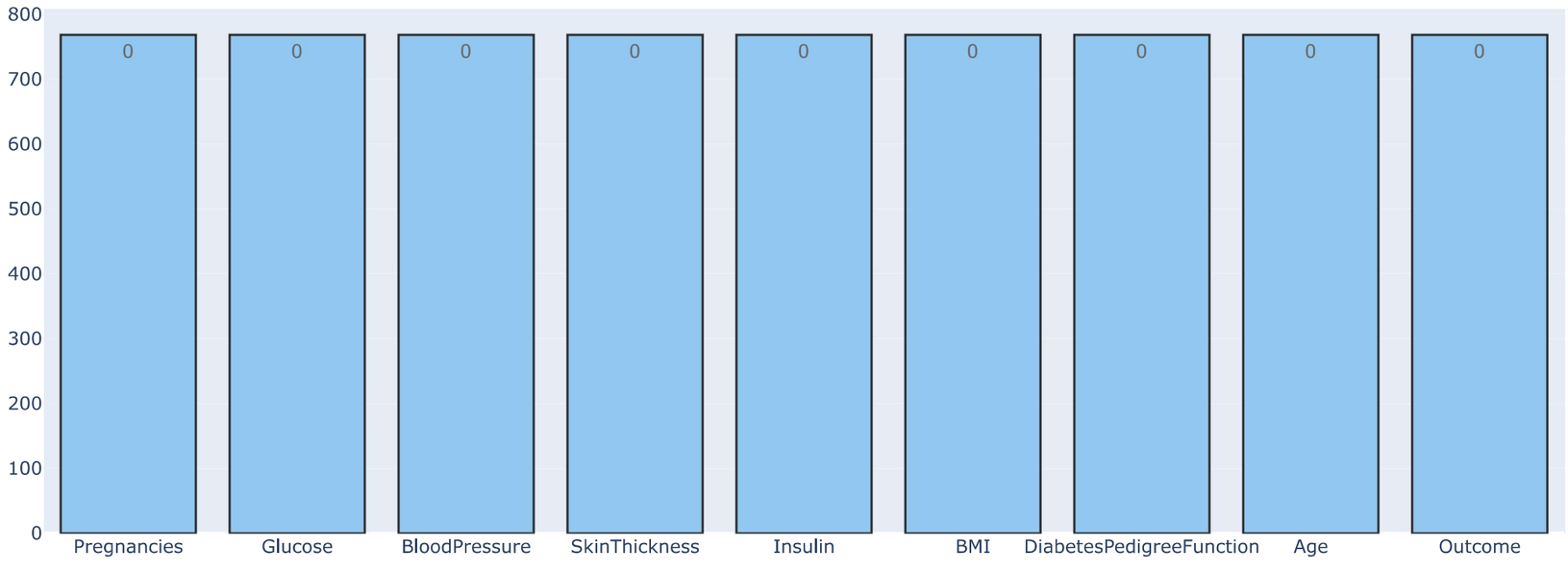
In [33]:
```
#plot distribution
plot_distribution('Age', 0)
plot_distribution('Pregnancies', 0)
plot_distribution('DiabetesPedigreeFunction', 0)
```


Age
0.1
0.05
0
20
30
40
50
60
70
80
healthy
diabetic


Pregnancies
0.2
0.15
0.1
0.05
0
0
2
4
6
8
10
12
14
16
18
healthy
diabetic



In [34]:
```
missing_plot(data, 'Outcome')
```



All features are complete. then we can create new features

4. New features (16) and EDA

Define 3 plots functions

In [35]:
```
def plot_feat1_feat2(feat1, feat2) :
    D = data[(data['Outcome'] != 0)]
    H = data[(data['Outcome'] == 0)]
    trace0 = go.Scatter(
        x = D[feat1],
        y = D[feat2],
        name = 'diabetic',
        mode = 'markers',
        marker = dict(color = '#FFD700',
            line = dict(
                width = 1)))

    trace1 = go.Scatter(
        x = H[feat1],
        y = H[feat2],
        name = 'healthy',
        mode = 'markers',
```

```
        marker = dict(color = '#7EC0EE',
            line = dict(
                width = 1)))

    layout = dict(title = feat1 +" "+"vs"+" "+ feat2,
                  yaxis = dict(title = feat2,zeroline = False),
                  xaxis = dict(title = feat1, zeroline = False)
                 )

    plots = [trace0, trace1]

    fig = dict(data = plots, layout=layout)
    py.iplot(fig)
```

In [36]:

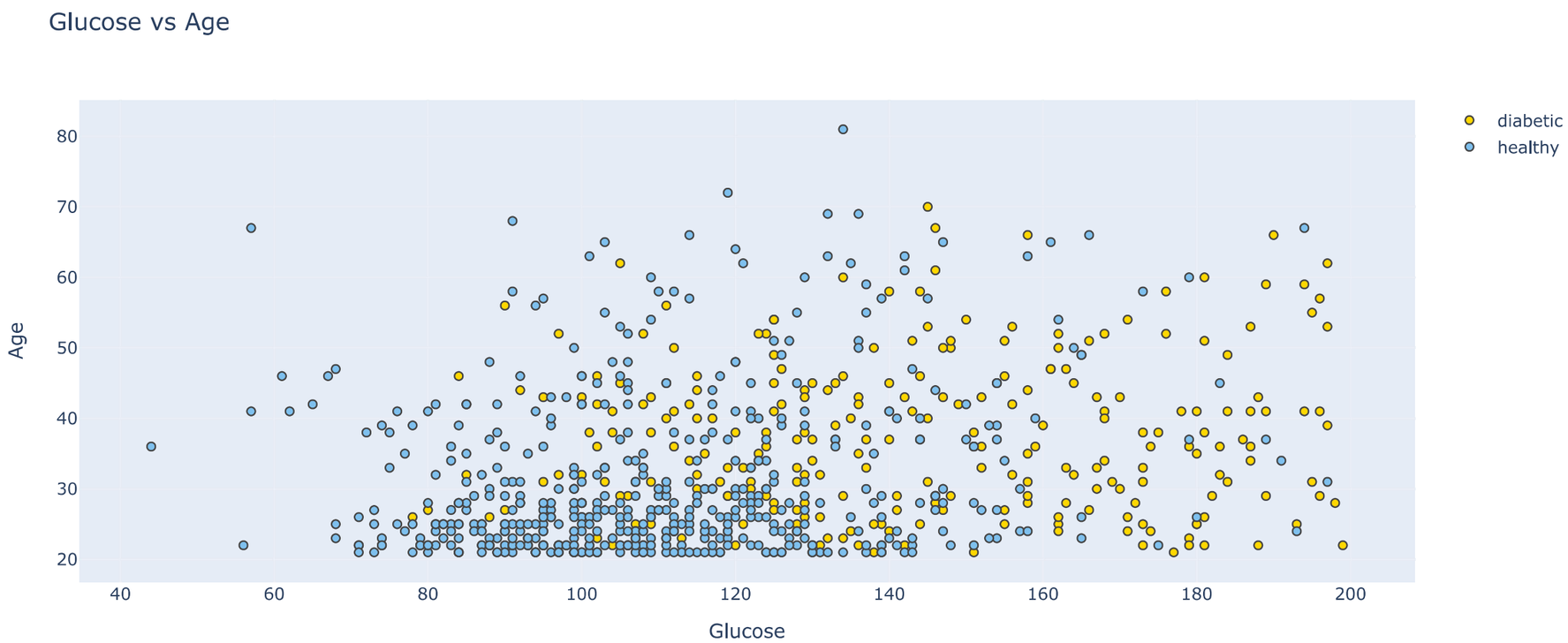
```
def barplot(var_select, sub) :
    tmp1 = data[(data['Outcome'] != 0)]
    tmp2 = data[(data['Outcome'] == 0)]
    tmp3 = pd.DataFrame(pd.crosstab(data[var_select],data['Outcome']), )
    tmp3['% diabetic'] = tmp3[1] / (tmp3[1] + tmp3[0]) * 100

    color=['lightskyblue','gold' ]
    trace1 = go.Bar(
        x=tmp1[var_select].value_counts().keys().tolist(),
        y=tmp1[var_select].value_counts().values.tolist(),
        text=tmp1[var_select].value_counts().values.tolist(),
        textposition = 'auto',
        name='diabetic',opacity = 0.8, marker=dict(
        color='gold',
        line=dict(color='#000000',width=1)))

    trace2 = go.Bar(
        x=tmp2[var_select].value_counts().keys().tolist(),
        y=tmp2[var_select].value_counts().values.tolist(),
        text=tmp2[var_select].value_counts().values.tolist(),
        textposition = 'auto',
        name='healthy', opacity = 0.8, marker=dict(
        color='lightskyblue',
        line=dict(color='#000000',width=1)))

    trace3 =  go.Scatter(
        x=tmp3.index,
        y=tmp3['% diabetic'],
        yaxis = 'y2',
        name='% diabetic', opacity = 0.6, marker=dict(
        color='black',
        line=dict(color='#000000',width=0.5
        )))

    layout = dict(title =  str(var_select)+' '+(sub),
              xaxis=dict(),
              yaxis=dict(title= 'Count'),
              yaxis2=dict(range= [-0, 75],
                          overlaying= 'y',
                          anchor= 'x',
                          side= 'right',
                          zeroline=False,
                          showgrid= False,
                          title= '% diabetic'
```

```
                        ))

    fig = go.Figure(data=[trace1, trace2, trace3], layout=layout)
    py.iplot(fig)
```

In [37]:
```
# Define pie plot to visualize each variable repartition vs target modalities : Survived or Died (train)
def plot_pie(var_select, sub) :
    D = data[(data['Outcome'] != 0)]
    H = data[(data['Outcome'] == 0)]

    col =['Silver', 'mediumturquoise','#CF5C36','lightblue','magenta', '#FF5D73','#F2D7EE','mediumturquoise']

    trace1 = go.Pie(values  = D[var_select].value_counts().values.tolist(),
                    labels  = D[var_select].value_counts().keys().tolist(),
                    textfont=dict(size=15), opacity = 0.8,
                    hole = 0.5,
                    hoverinfo = "label+percent+name",
                    domain  = dict(x = [.0,.48]),
                    name    = "Diabetic",
                    marker  = dict(colors = col, line = dict(width = 1.5)))
    trace2 = go.Pie(values  = H[var_select].value_counts().values.tolist(),
                    labels  = H[var_select].value_counts().keys().tolist(),
                    textfont=dict(size=15), opacity = 0.8,
                    hole = 0.5,
                    hoverinfo = "label+percent+name",
                    marker  = dict(line = dict(width = 1.5)),
                    domain  = dict(x = [.52,1]),
                    name    = "Healthy" )

    layout = go.Layout(dict(title = var_select + " distribution by target <br>"+(sub),
                            annotations = [ dict(text = "Diabetic"+" : "+"268",
                                                 font = dict(size = 13),
                                                 showarrow = False,
                                                 x = .22, y = -0.1),
                                            dict(text = "Healthy"+" : "+"500",
                                                 font = dict(size = 13),
                                                 showarrow = False,
                                                 x = .8,y = -.1)]))


    fig  = go.Figure(data = [trace1,trace2],layout = layout)
    py.iplot(fig)
```

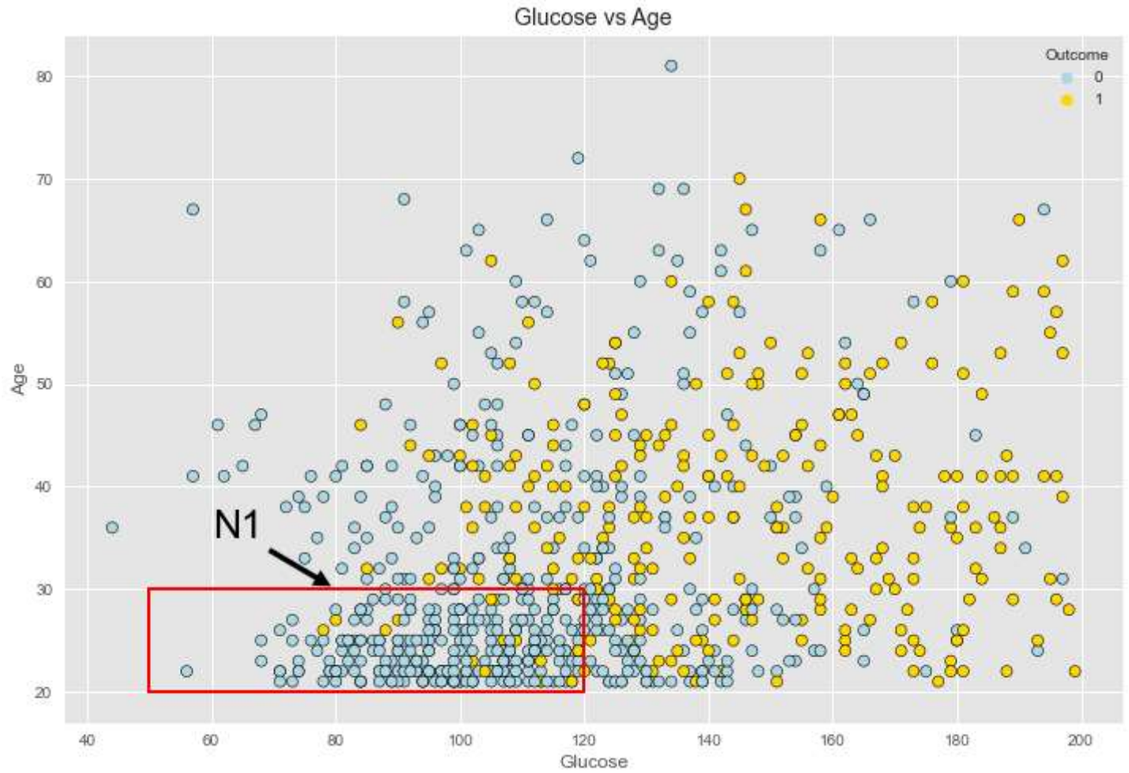
- **Glucose and Age**

In [38]:
```
plot_feat1_feat2('Glucose','Age')
```



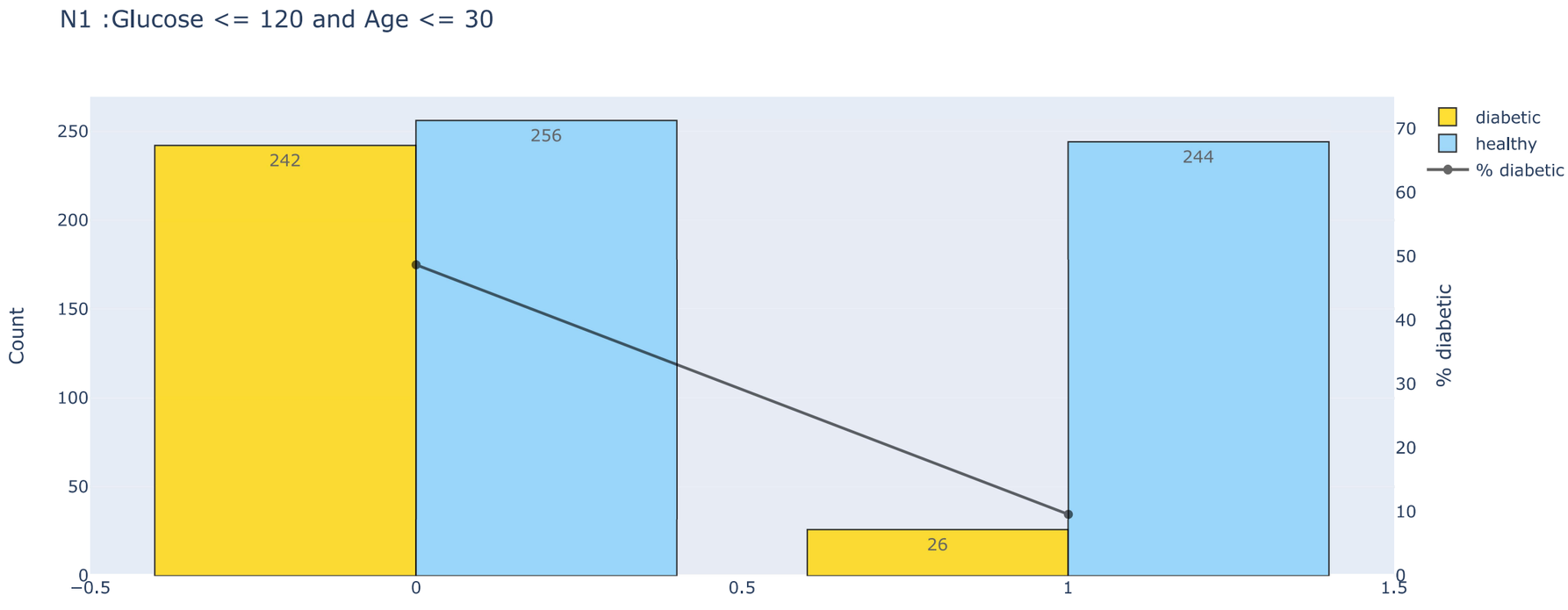
Healthy persons are concentrate with an age <= 30 and glucose <= 120

In [39]:
```
palette ={0 : 'lightblue', 1 : 'gold'}
edgecolor = 'black'

fig = plt.figure(figsize=(12,8))

ax1 = sns.scatterplot(x = data['Glucose'], y = data['Age'], hue = "Outcome",
                      data = data, palette = palette, edgecolor=edgecolor)

plt.annotate('N1', size=25, color='black', xy=(80, 30), xytext=(60, 35),
             arrowprops=dict(facecolor='black', shrink=0.05),
             )
plt.plot([50, 120], [30, 30], linewidth=2, color = 'red')
plt.plot([120, 120], [20, 30], linewidth=2, color = 'red')
plt.plot([50, 120], [20, 20], linewidth=2, color = 'red')
plt.plot([50, 50], [20, 30], linewidth=2, color = 'red')
plt.title('Glucose vs Age')
plt.show()
```

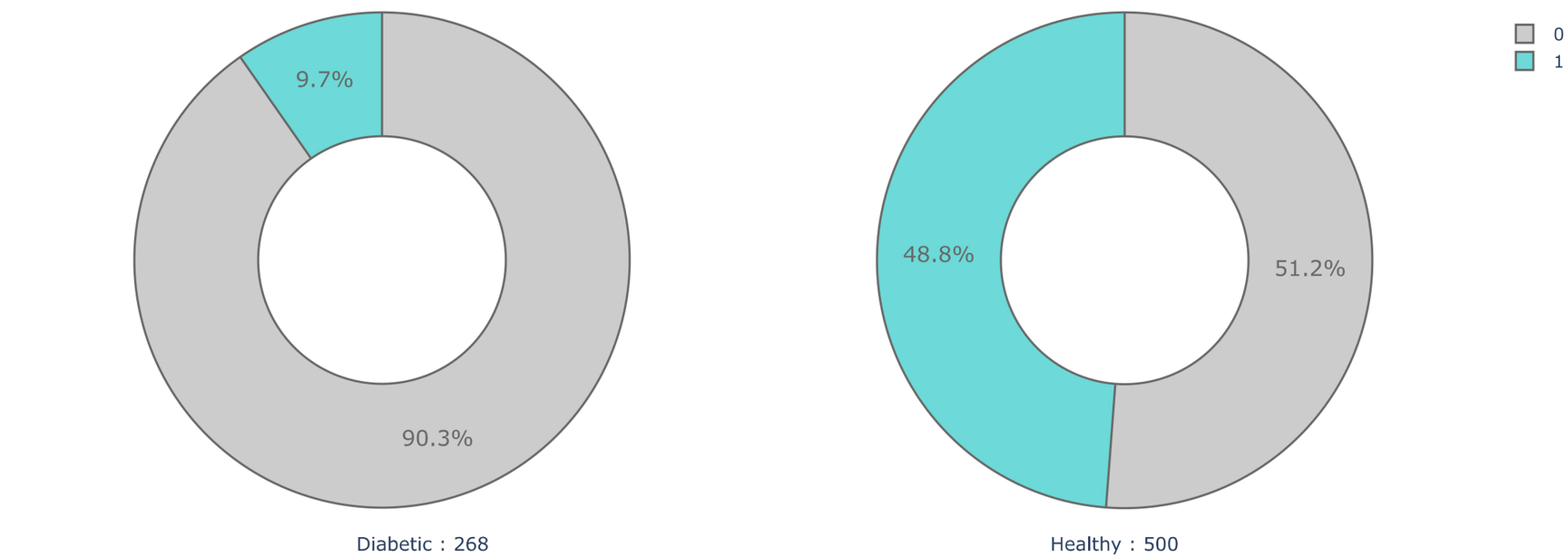


In [40]:
```
data.loc[:,'N1']=0
data.loc[(data['Age']<=30) & (data['Glucose']<=120),'N1']=1
```

In [41]:
```
barplot('N1', ':Glucose <= 120 and Age <= 30')
```



In [42]:
```
plot_pie('N1', '(Glucose <= 120 and Age <= 30)')
```



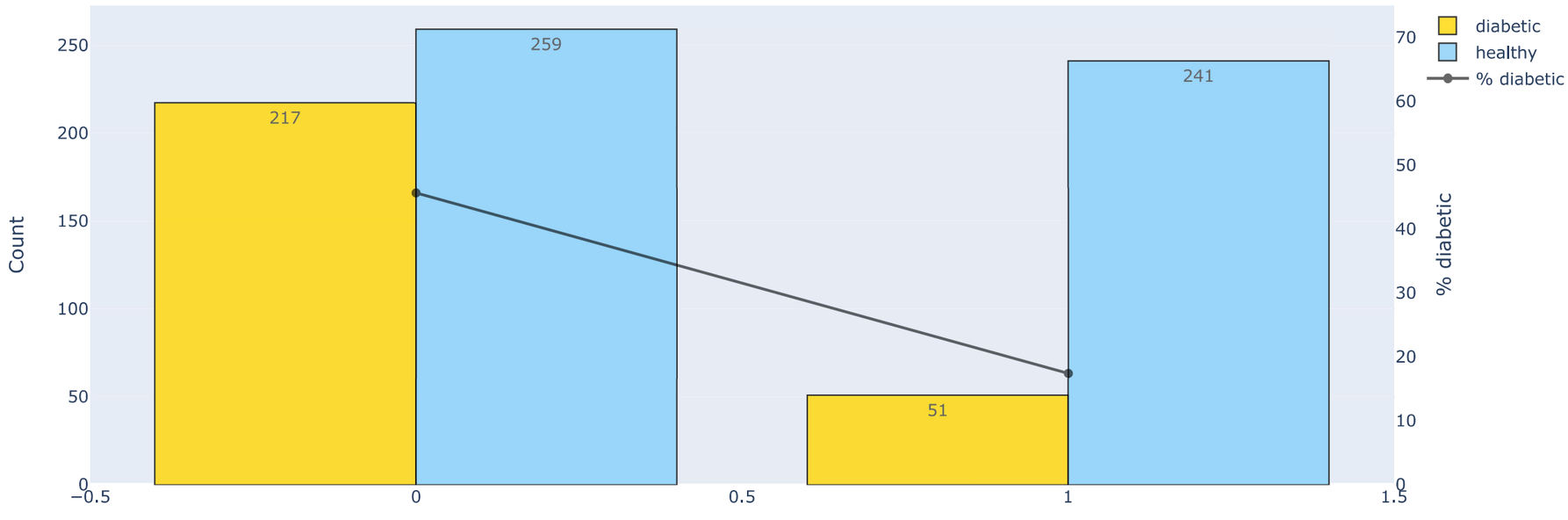
- **BMI**

According to Wikipedia, "The body mass index (BMI) or Quetelet index is a value derived from an individual's mass (weight) and height. BMI is calculated as the body mass divided by the square of the body height and is universally expressed in units of kg/m^2, derived from mass in kilograms and height in meters."

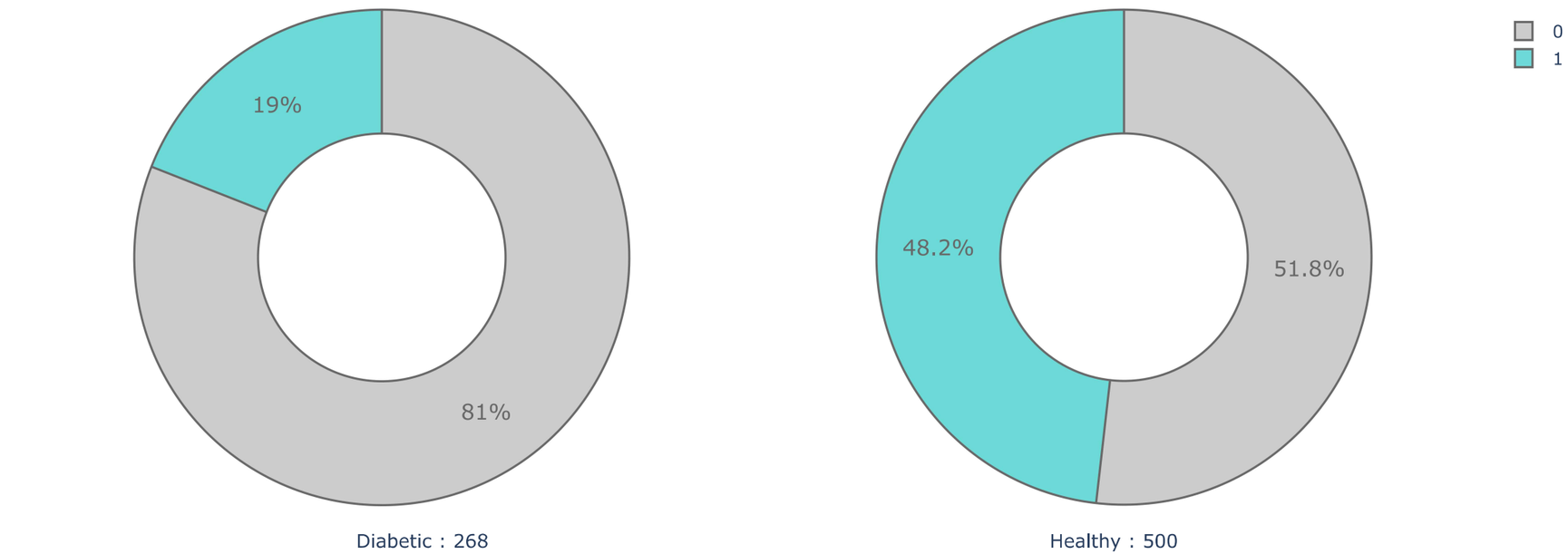
The BMI value of 30 kg/m^2 represents the threshold for obesity.

In [43]:
```
data.loc[:,'N2']=0
data.loc[(data['BMI']<=30),'N2']=1
```

In [44]:
```
barplot('N2', ': BMI <= 30')
```

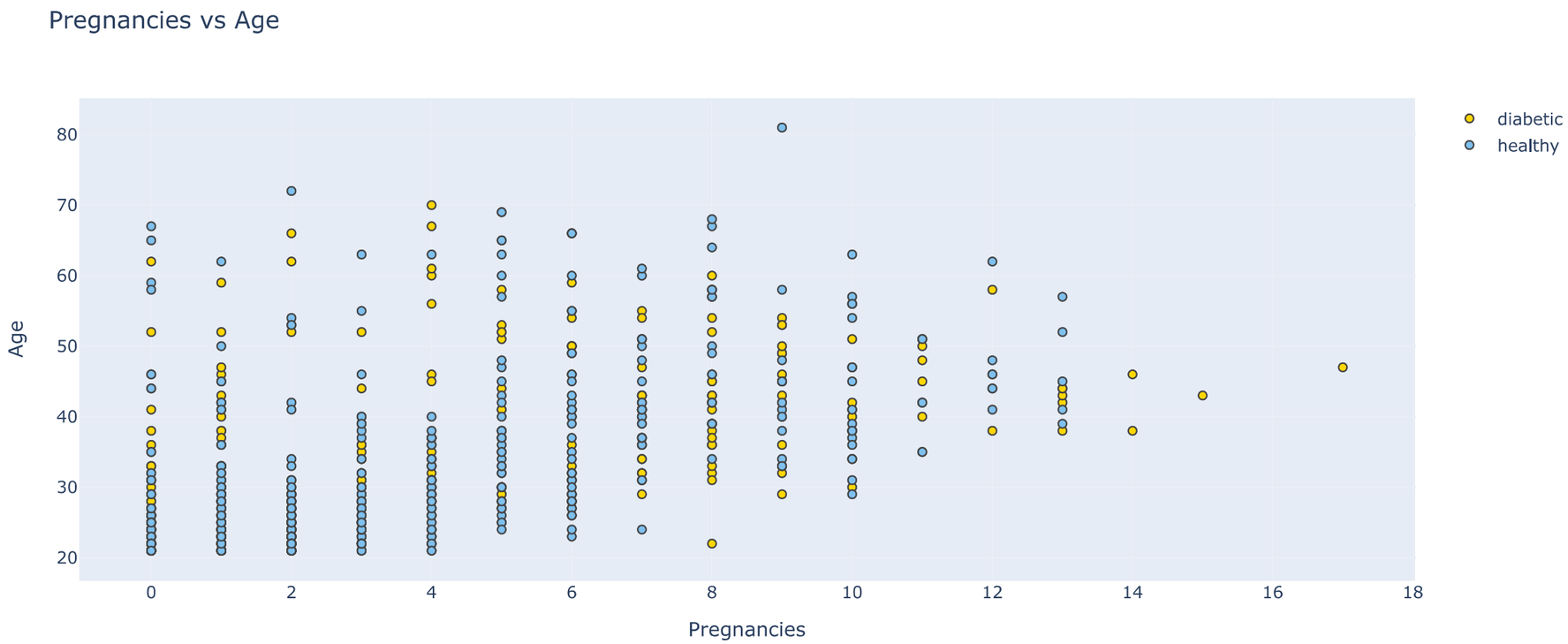


In [45]:
```
plot_pie('N2', 'BMI <= 30')
```



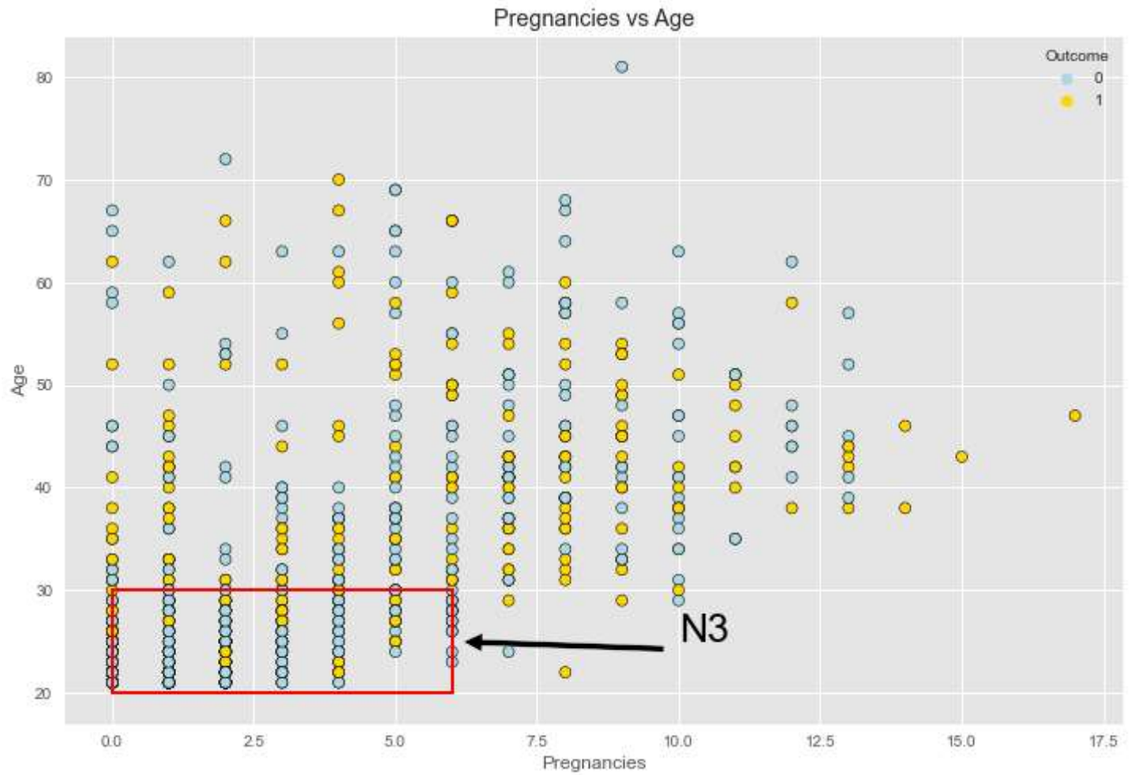
- **Pregnancies and Age**

In [46]:
```
plot_feat1_feat2('Pregnancies','Age')
```



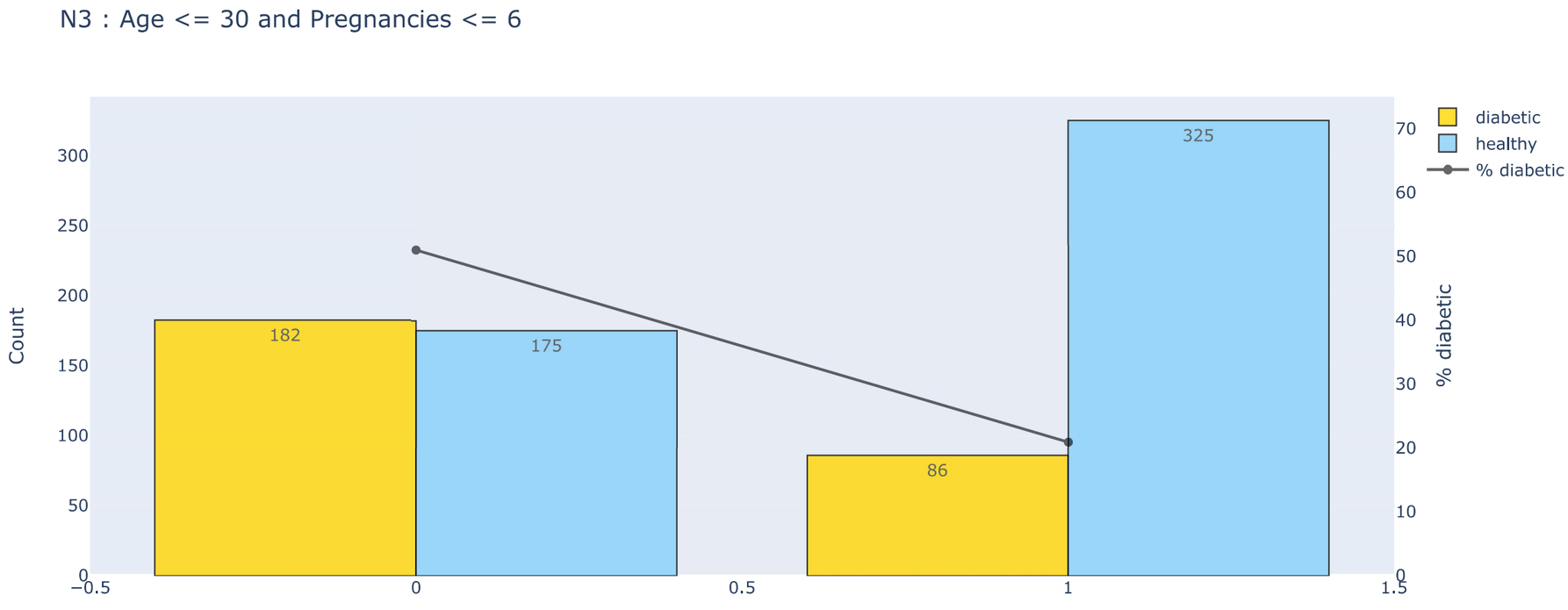
In [47]:

```
palette ={0 : 'lightblue', 1 : 'gold'}
edgecolor = 'black'

fig = plt.figure(figsize=(12,8))

ax1 = sns.scatterplot(x = data['Pregnancies'], y = data['Age'], hue = "Outcome",
                      data = data, palette = palette, edgecolor=edgecolor)

plt.annotate('N3', size=25, color='black', xy=(6, 25), xytext=(10, 25),
             arrowprops=dict(facecolor='black', shrink=0.05),
             )
plt.plot([0, 6], [30, 30], linewidth=2, color = 'red')
plt.plot([6, 6], [20, 30], linewidth=2, color = 'red')
plt.plot([0, 6], [20, 20], linewidth=2, color = 'red')
plt.plot([0, 0], [20, 30], linewidth=2, color = 'red')
plt.title('Pregnancies vs Age')
plt.show()
```

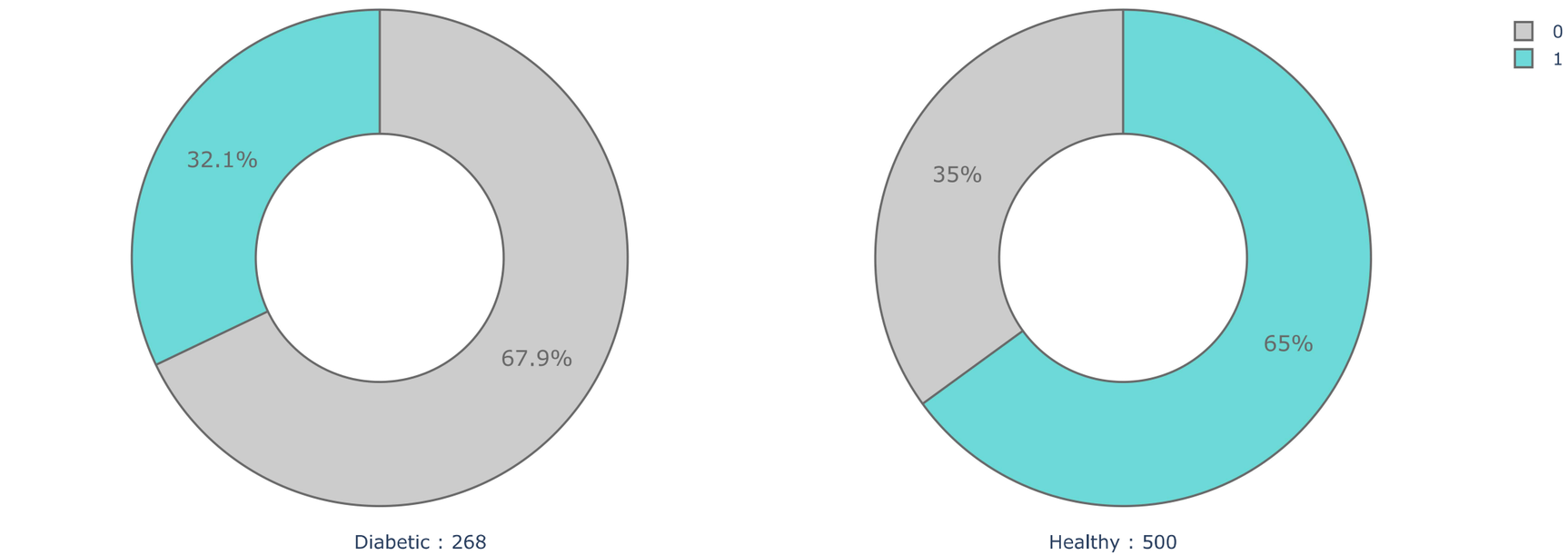


In [48]:
```
data.loc[:,'N3']=0
data.loc[(data['Age']<=30) & (data['Pregnancies']<=6),'N3']=1
```

In [49]:
```
barplot('N3', ': Age <= 30 and Pregnancies <= 6')
```

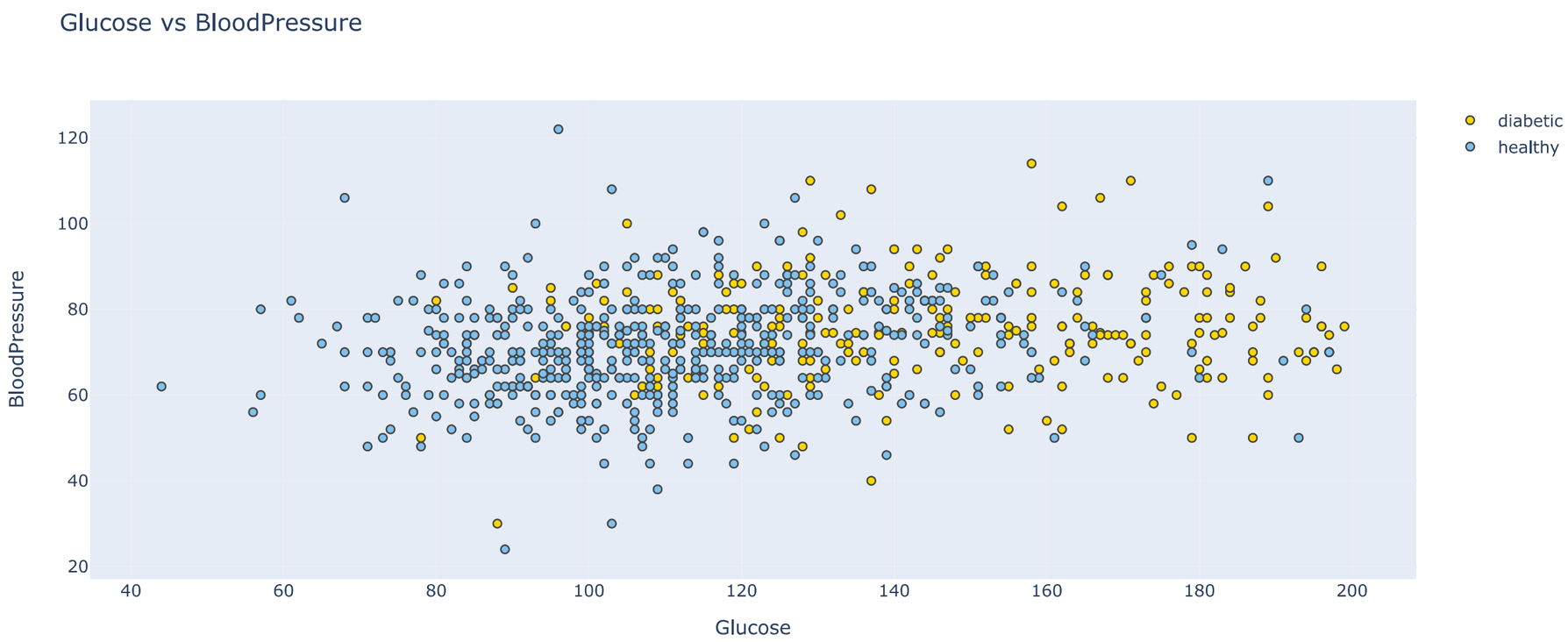


In [50]:
```
plot_pie('N3', 'Age <= 30 and Pregnancies <= 6')
```



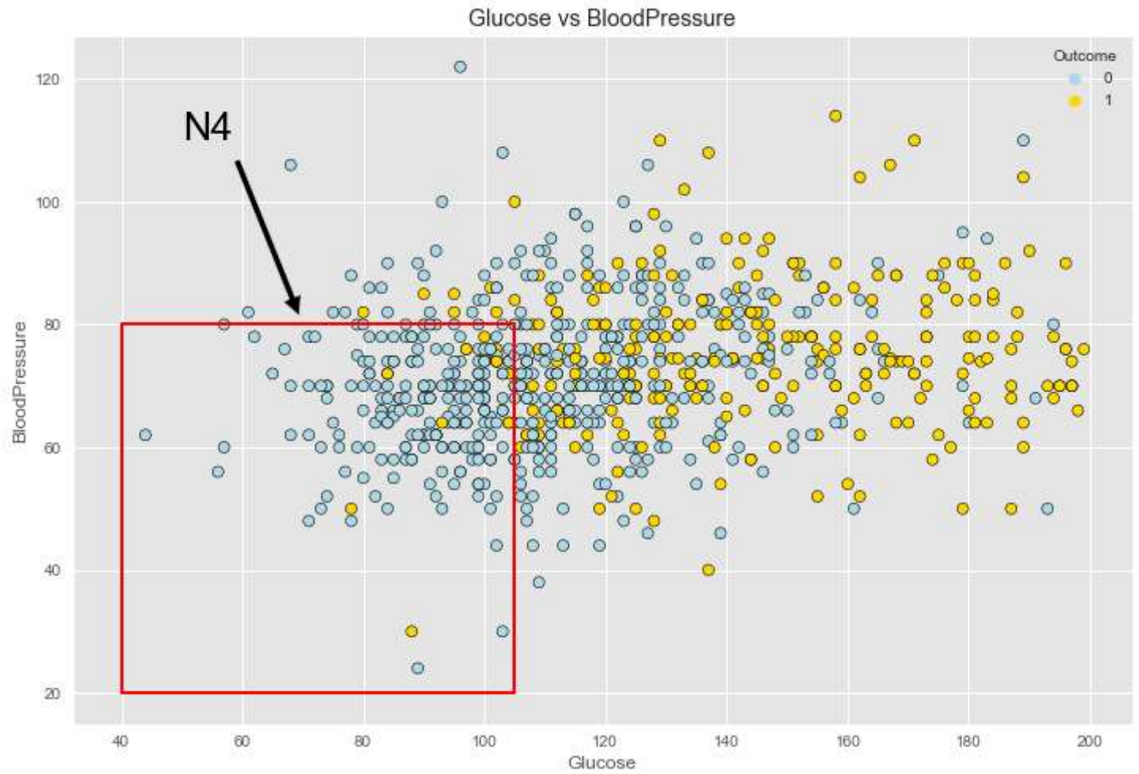
- **Glucose and BloodPressure**

In [51]:
```
plot_feat1_feat2('Glucose','BloodPressure')
```



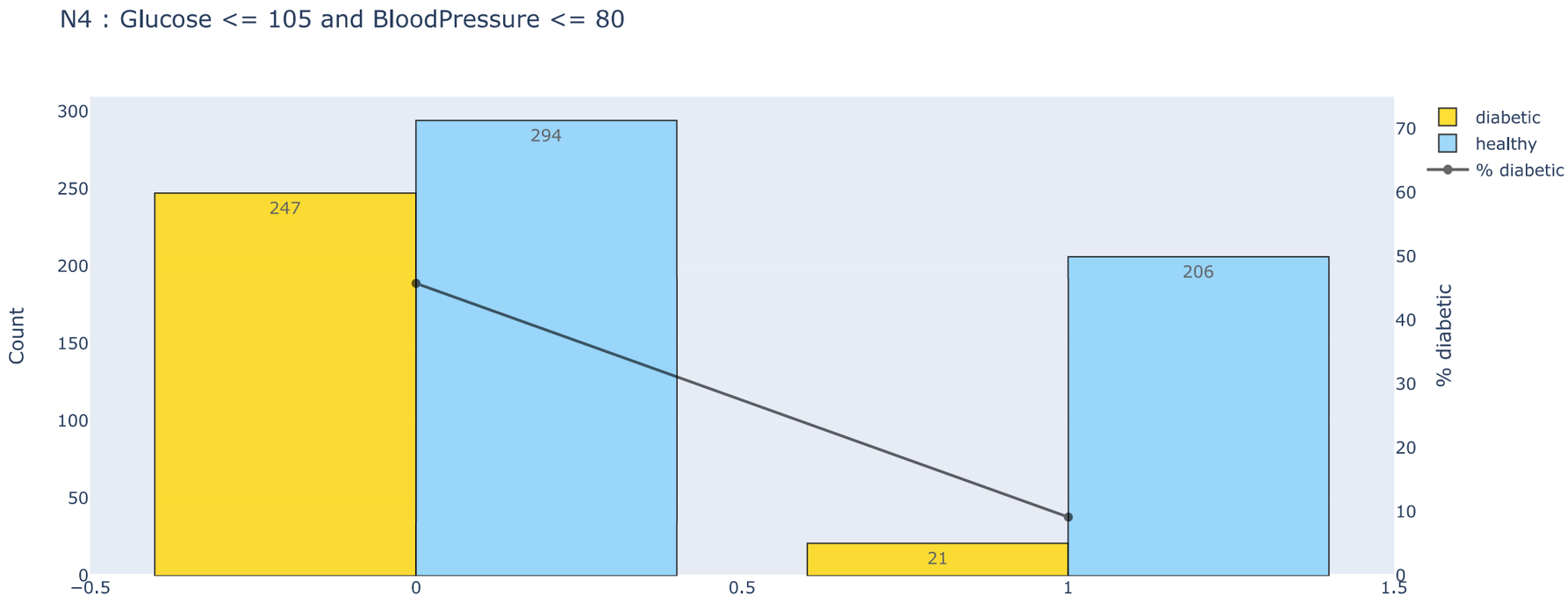
Healthy persons are concentrate with an blood pressure <= 80 and glucose <= 105

In [52]:
```
palette ={0 : 'lightblue', 1 : 'gold'}
edgecolor = 'black'

fig = plt.figure(figsize=(12,8))

ax1 = sns.scatterplot(x = data['Glucose'], y = data['BloodPressure'], hue = "Outcome",
                      data = data, palette = palette, edgecolor=edgecolor)

plt.annotate('N4', size=25, color='black', xy=(70, 80), xytext=(50, 110),
             arrowprops=dict(facecolor='black', shrink=0.05),
             )
plt.plot([40, 105], [80, 80], linewidth=2, color = 'red')
plt.plot([40, 40], [20, 80], linewidth=2, color = 'red')
plt.plot([40, 105], [20, 20], linewidth=2, color = 'red')
plt.plot([105, 105], [20, 80], linewidth=2, color = 'red')
plt.title('Glucose vs BloodPressure')
plt.show()
```

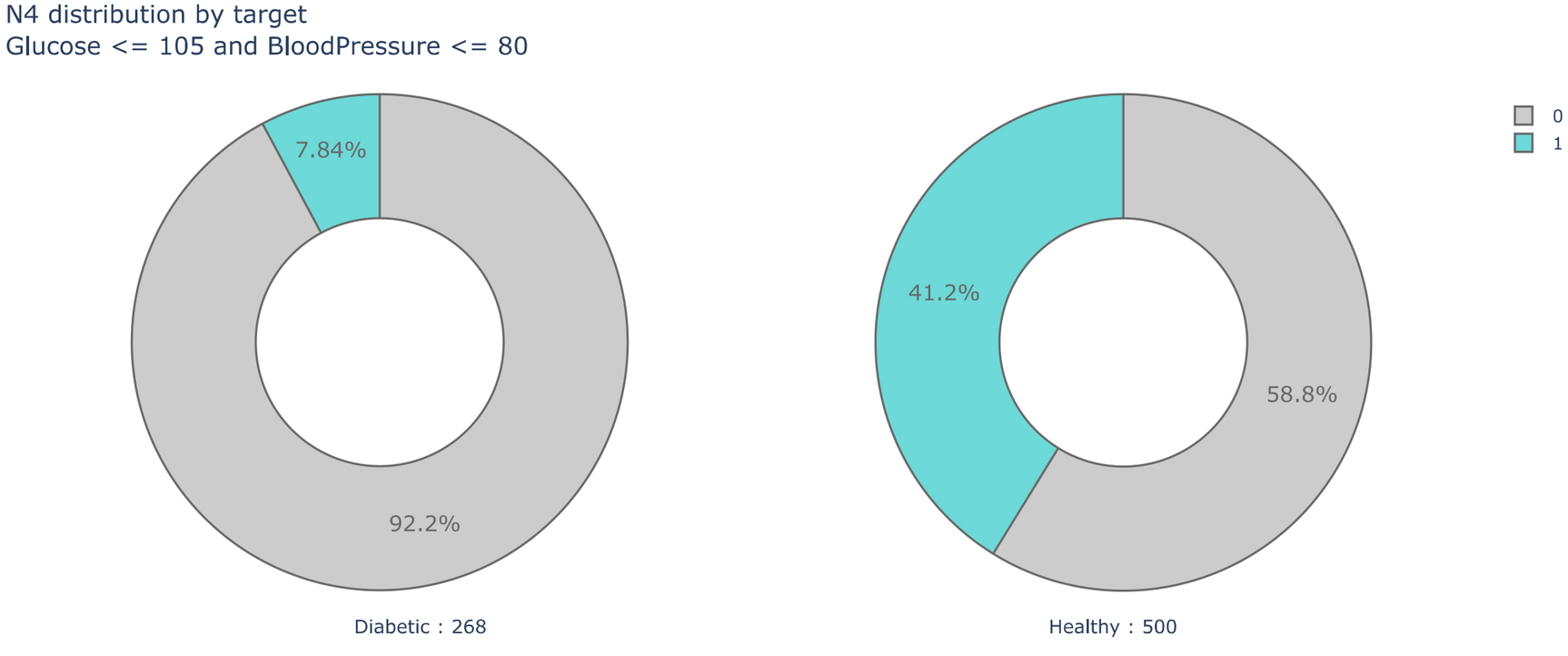


In [53]:
```
data.loc[:,'N4']=0
data.loc[(data['Glucose']<=105) & (data['BloodPressure']<=80),'N4']=1
```

In [54]:
```
barplot('N4', ': Glucose <= 105 and BloodPressure <= 80')
```



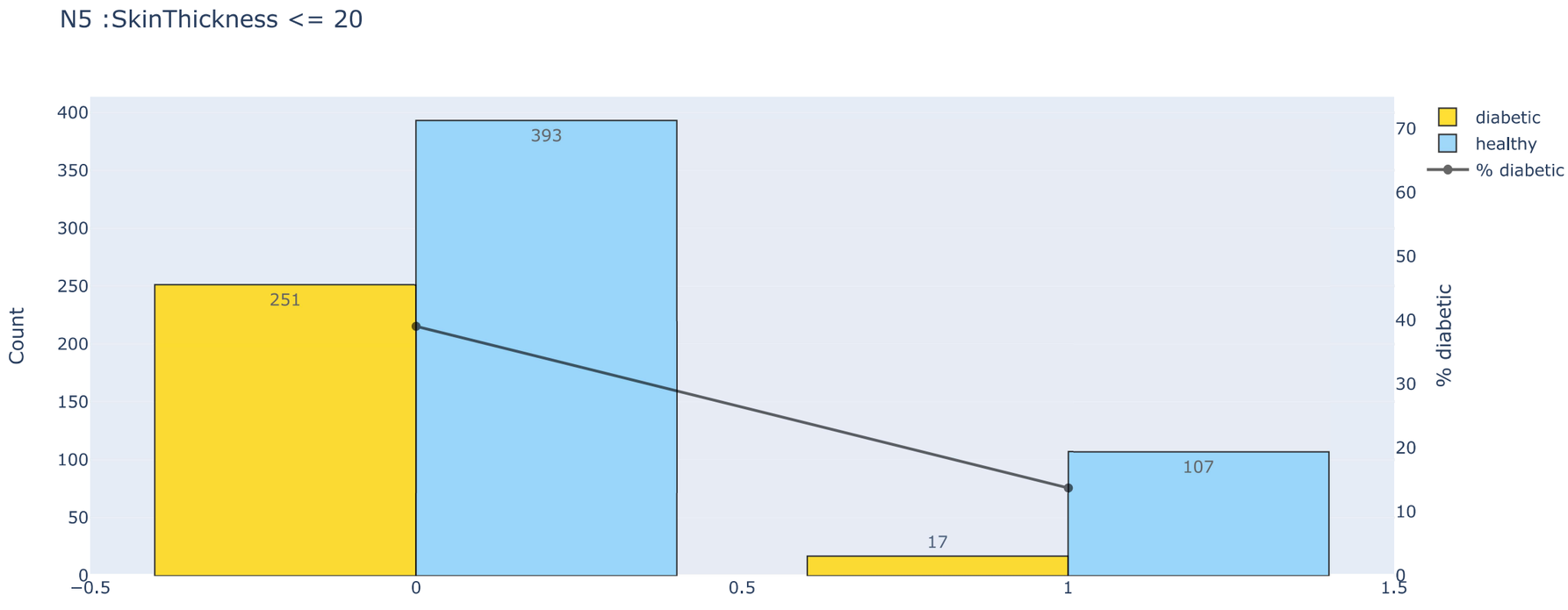
In [55]:
```
plot_pie('N4', 'Glucose <= 105 and BloodPressure <= 80')
```



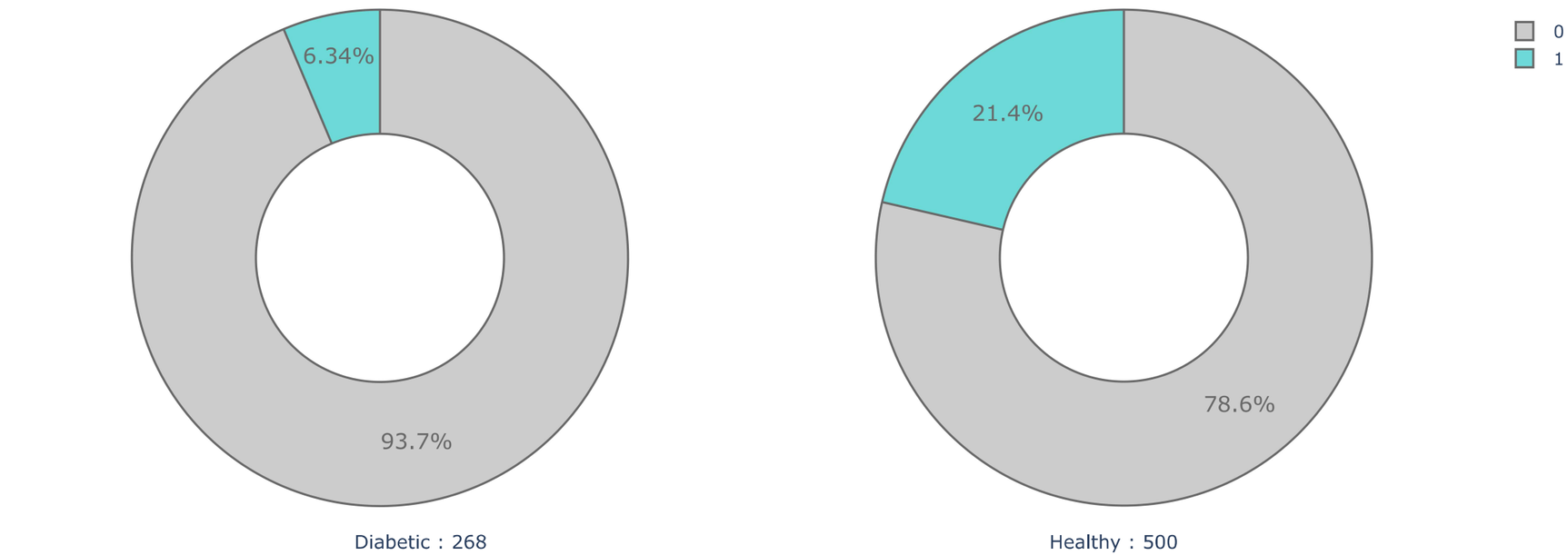
- **SkinThickness**

In [56]:
```
data.loc[:,'N5']=0
data.loc[(data['SkinThickness']<=20) ,'N5']=1
```

In [57]:
```
barplot('N5', ':SkinThickness <= 20')
```

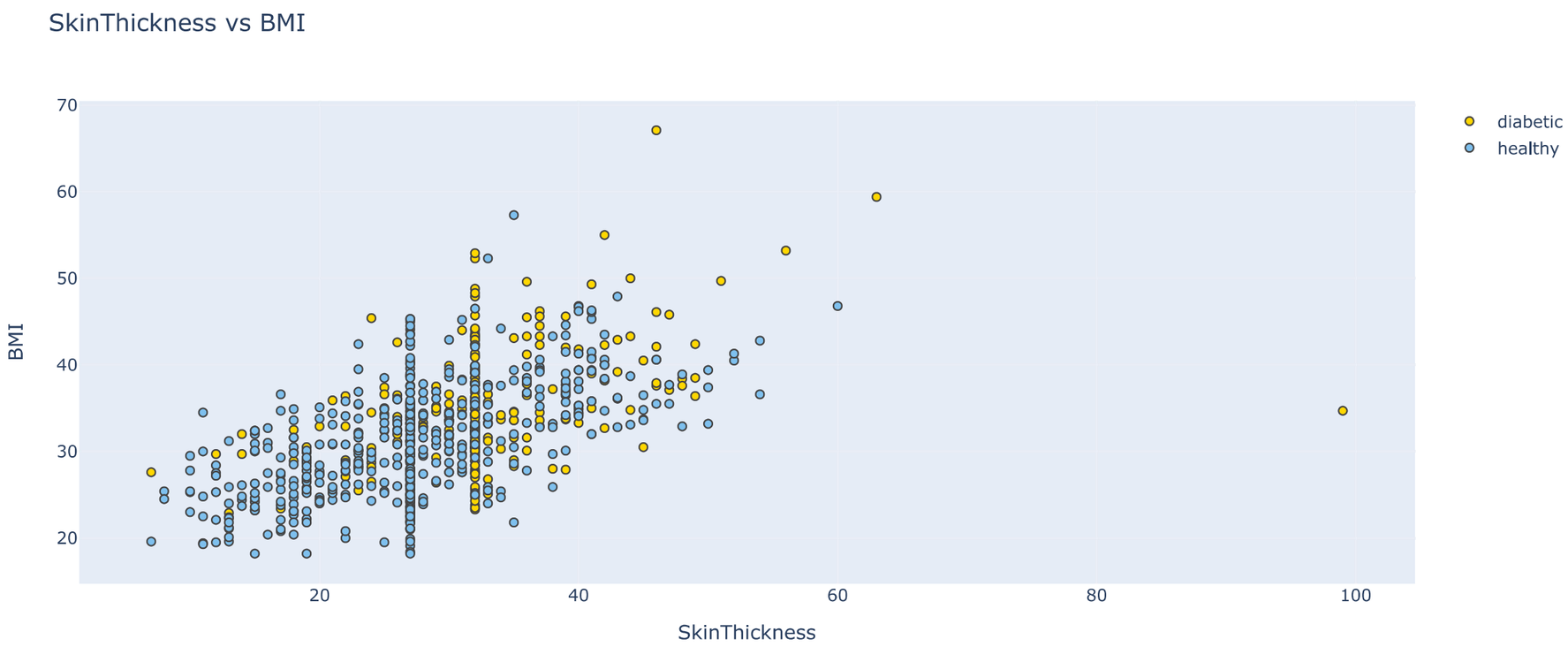


In [58]:
```
plot_pie('N5', 'SkinThickness <= 20')
```



- **SkinThickness and BMI**

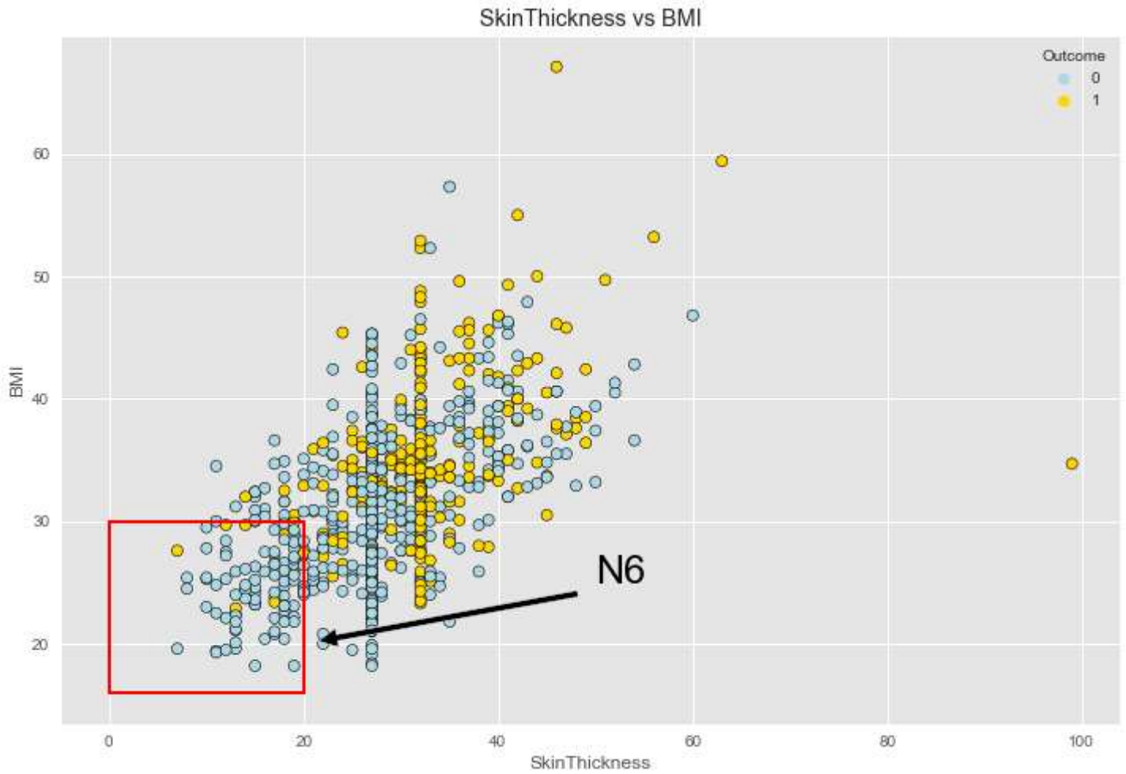
In [59]:
```
plot_feat1_feat2('SkinThickness','BMI')
```



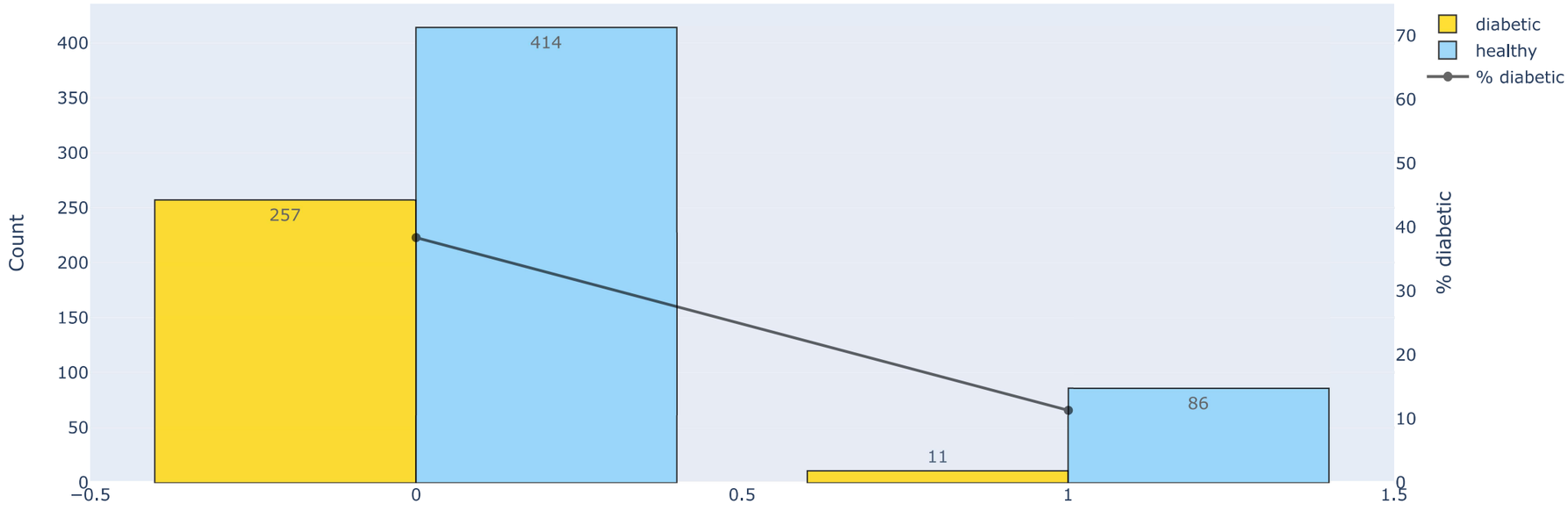
Healthy persons are concentrate with a BMI < 30 and skin thickness <= 20

In [60]:
```
data.loc[:,'N6']=0
data.loc[(data['BMI']<30) & (data['SkinThickness']<=20),'N6']=1
```

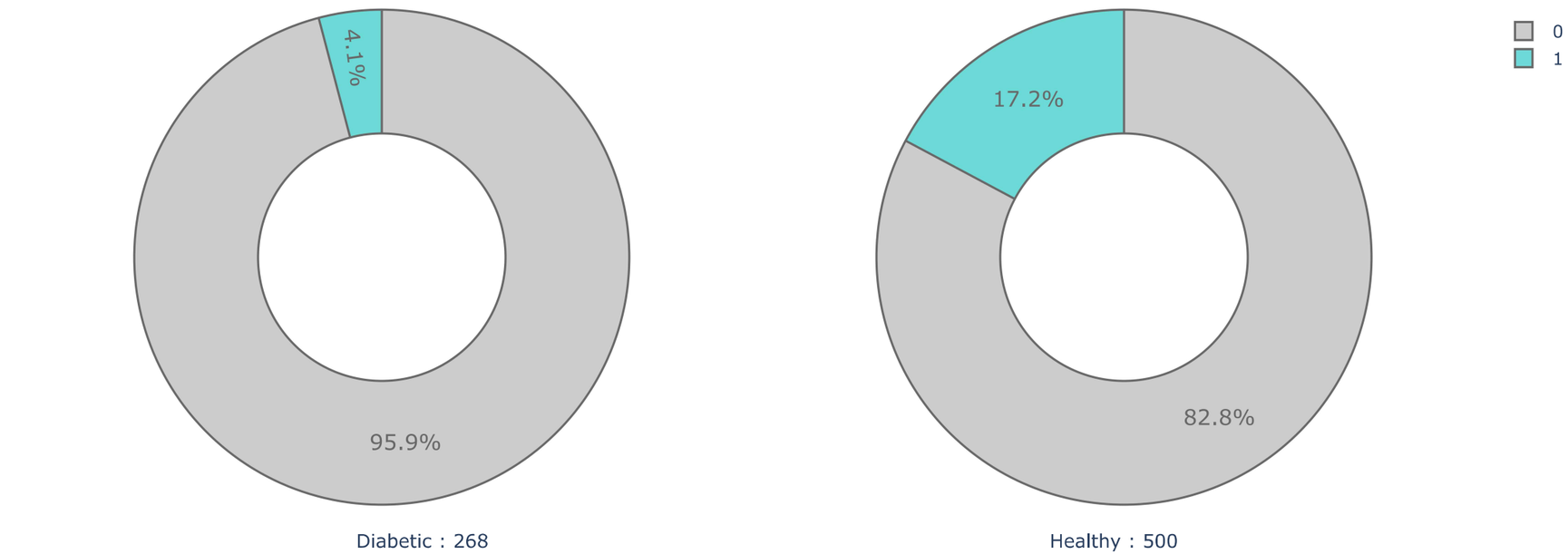
In [61]:
```
palette ={0 : 'lightblue', 1 : 'gold'}
edgecolor = 'black'

fig = plt.figure(figsize=(12,8))

ax1 = sns.scatterplot(x = data['SkinThickness'], y = data['BMI'], hue = "Outcome",
                      data = data, palette = palette, edgecolor=edgecolor)

plt.annotate('N6', size=25, color='black', xy=(20, 20), xytext=(50, 25),
             arrowprops=dict(facecolor='black', shrink=0.05),
             )
plt.plot([0, 20], [30, 30], linewidth=2, color = 'red')
plt.plot([0, 0], [16, 30], linewidth=2, color = 'red')
plt.plot([0, 20], [16, 16], linewidth=2, color = 'red')
plt.plot([20, 20], [16, 30], linewidth=2, color = 'red')
plt.title('SkinThickness vs BMI')
plt.show()
```



In [62]:
```
barplot('N6', ': BMI < 30 and SkinThickness <= 20')
```

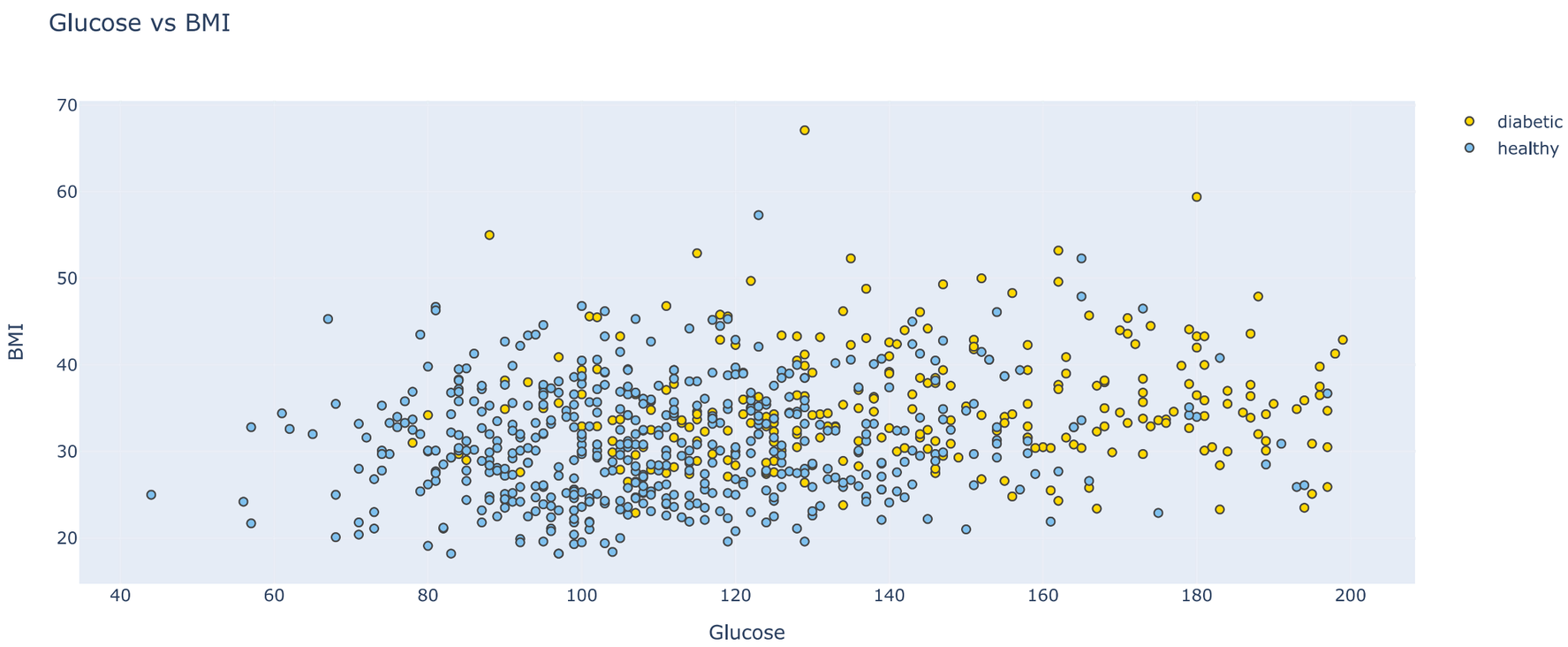


In [63]:
```
plot_pie('N6', 'BMI < 30 and SkinThickness <= 20')
```



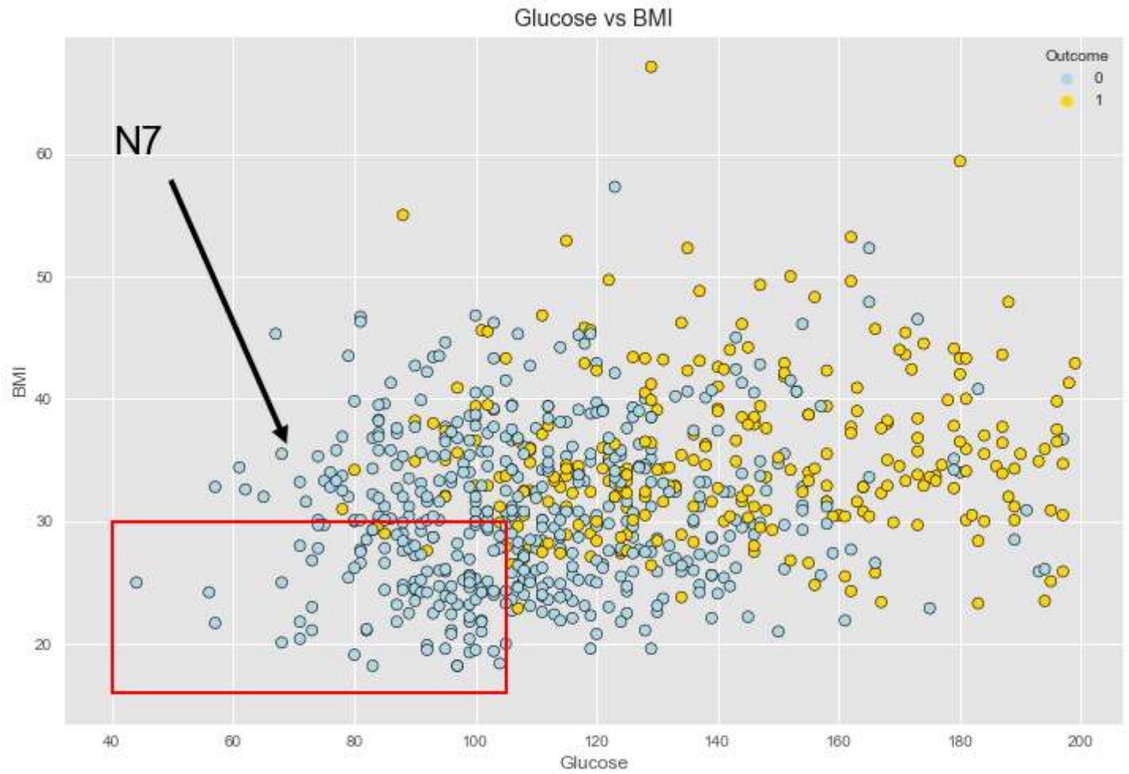
- **Glucose and BMI**

In [64]:
```
plot_feat1_feat2('Glucose','BMI')
```



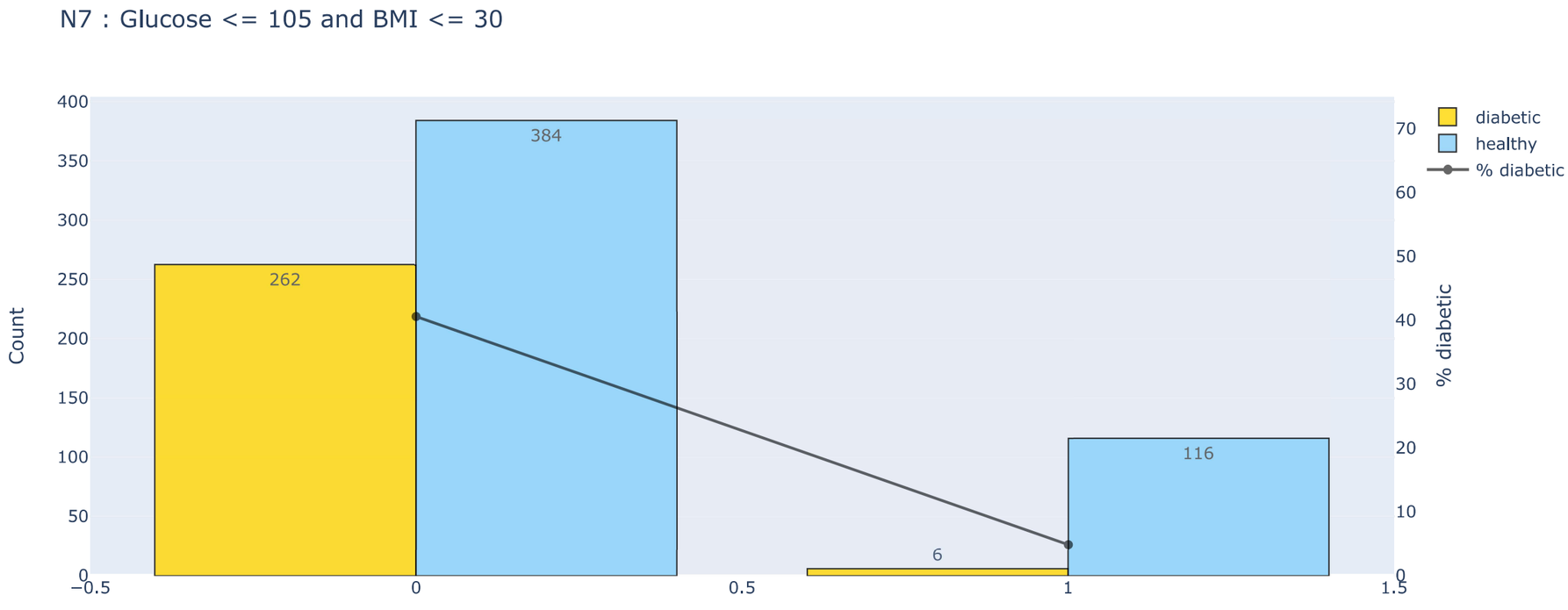
In [65]:

```
palette ={0 : 'lightblue', 1 : 'gold'}
edgecolor = 'black'

fig = plt.figure(figsize=(12,8))

ax1 = sns.scatterplot(x = data['Glucose'], y = data['BMI'], hue = "Outcome",
                      data = data, palette = palette, edgecolor=edgecolor)

plt.annotate('N7', size=25, color='black', xy=(70, 35), xytext=(40, 60),
             arrowprops=dict(facecolor='black', shrink=0.05),
             )
plt.plot([105, 105], [16, 30], linewidth=2, color = 'red')
plt.plot([40, 40], [16, 30], linewidth=2, color = 'red')
plt.plot([40, 105], [16, 16], linewidth=2, color = 'red')
plt.plot([40, 105], [30, 30], linewidth=2, color = 'red')
plt.title('Glucose vs BMI')
plt.show()
```

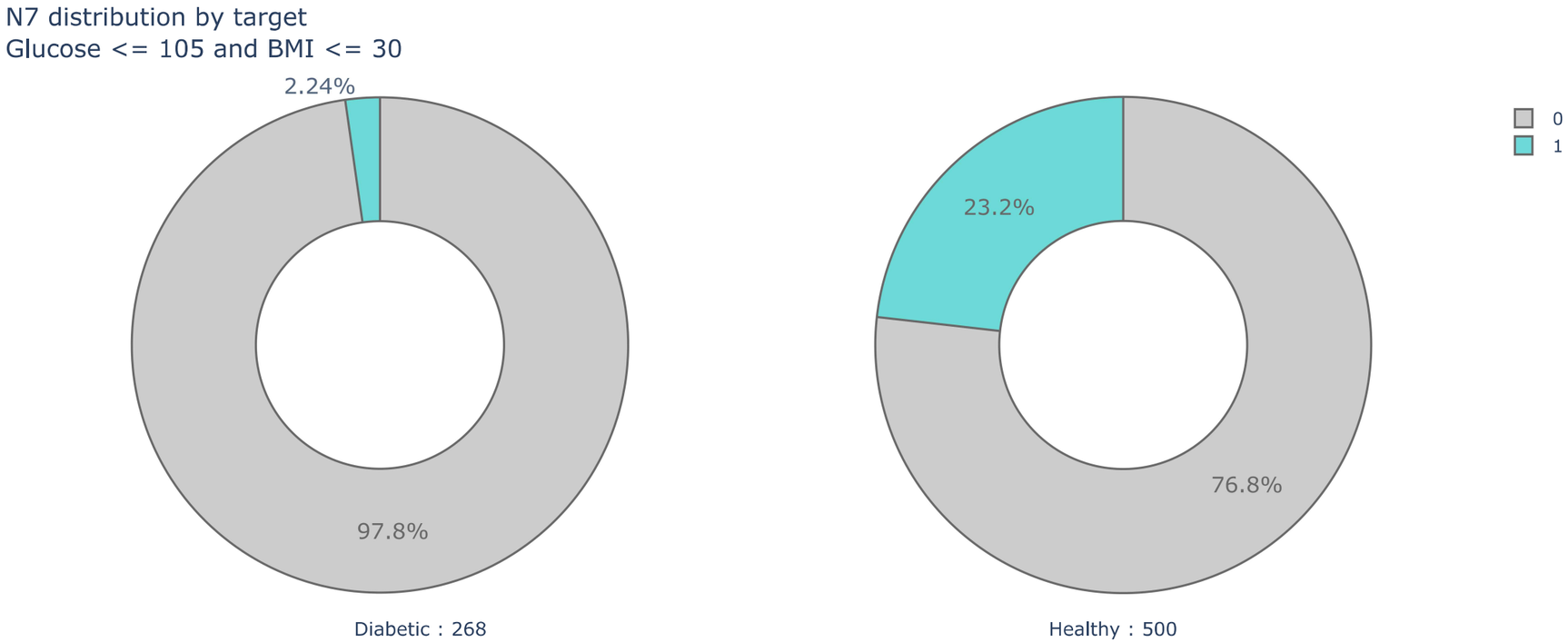


In [66]:
```
data.loc[:,'N7']=0
data.loc[(data['Glucose']<=105) & (data['BMI']<=30),'N7']=1
```

In [67]:
```
barplot('N7', ': Glucose <= 105 and BMI <= 30')
```

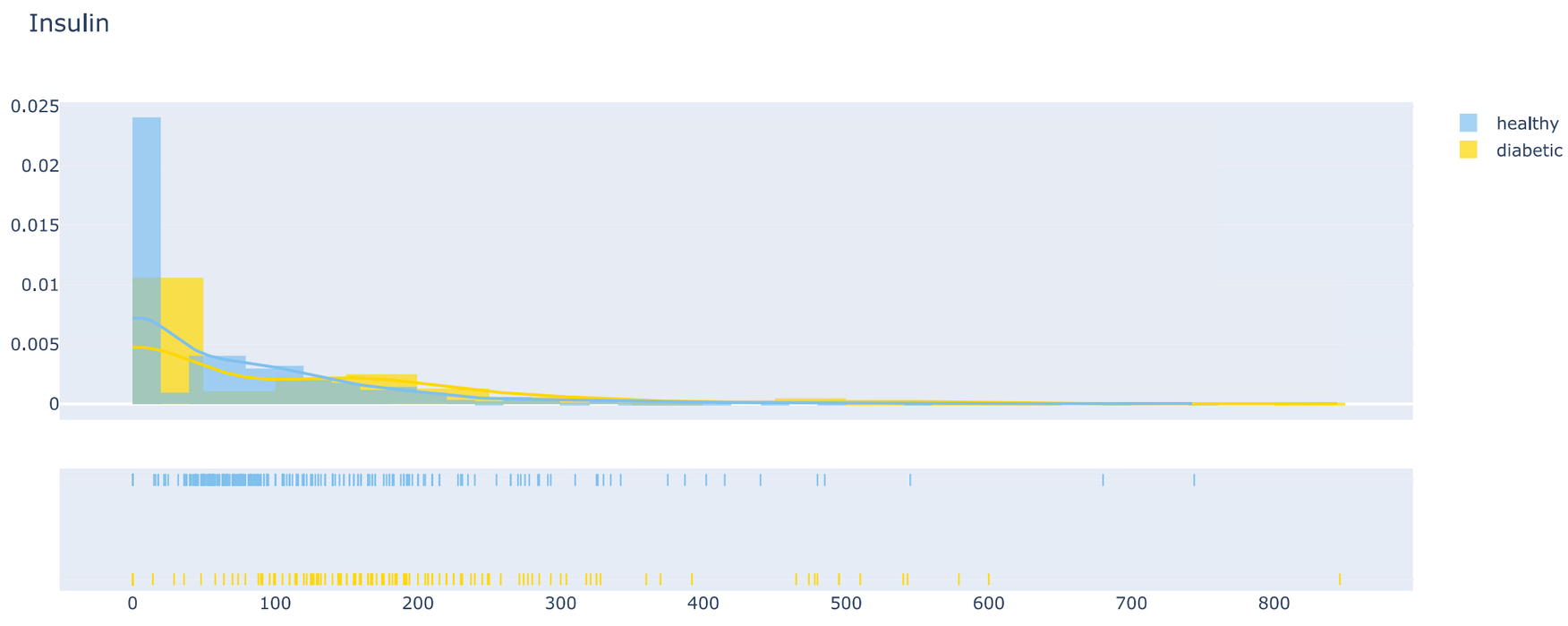


In [68]:
```
plot_pie('N7', 'Glucose <= 105 and BMI <= 30')
```



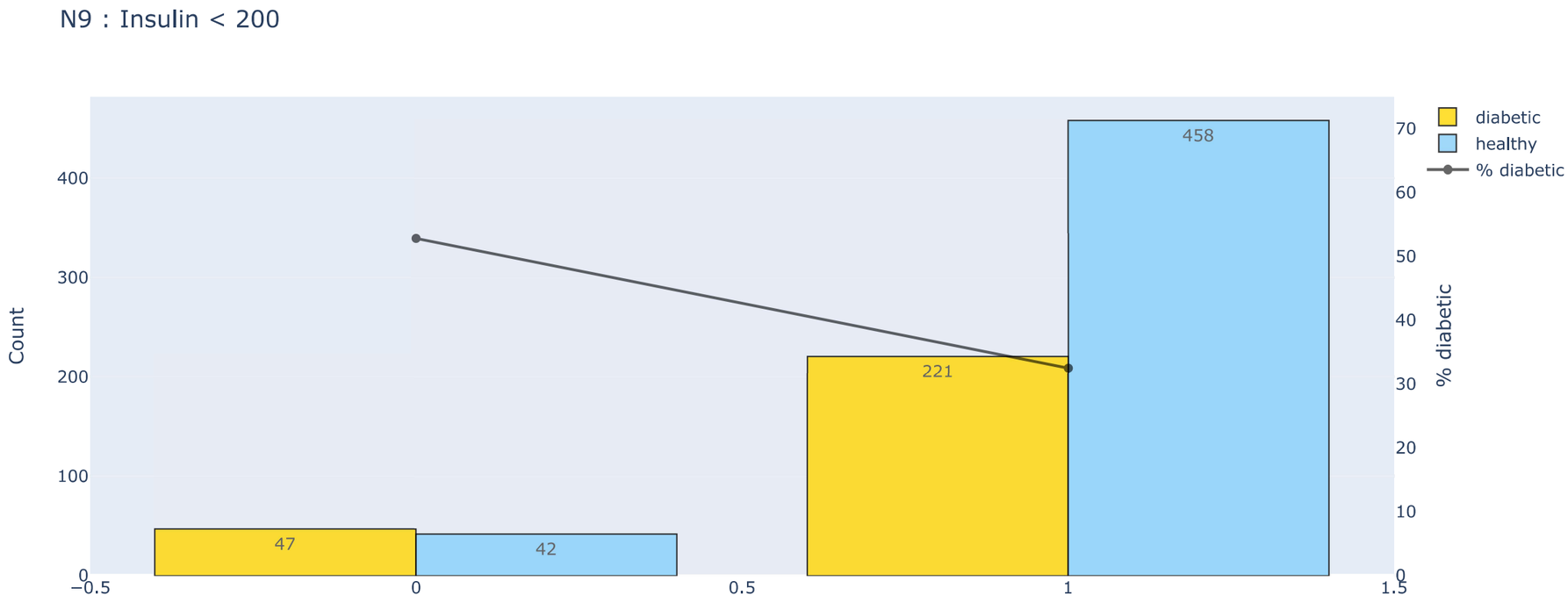
- **Insulin**

In [69]:
```
plot_distribution('Insulin', 0)
```

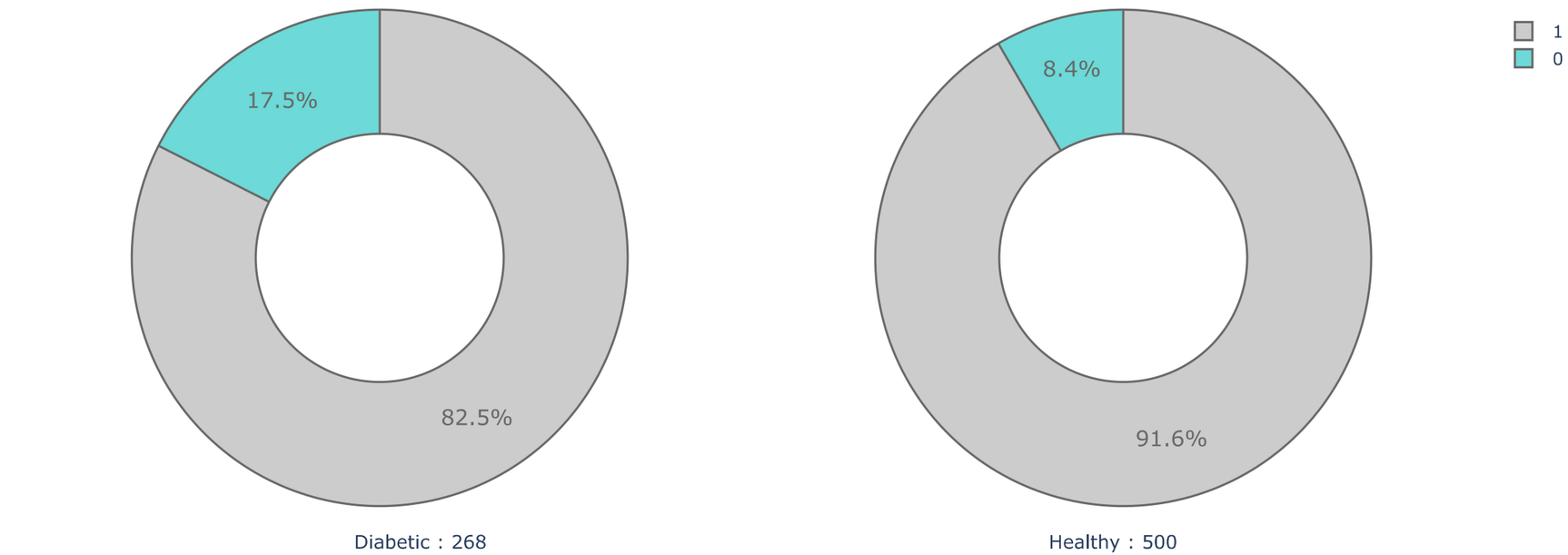


In [70]:
```
data.loc[:,'N9']=0
data.loc[(data['Insulin']<200),'N9']=1
```

In [71]:
```
barplot('N9', ': Insulin < 200')
```



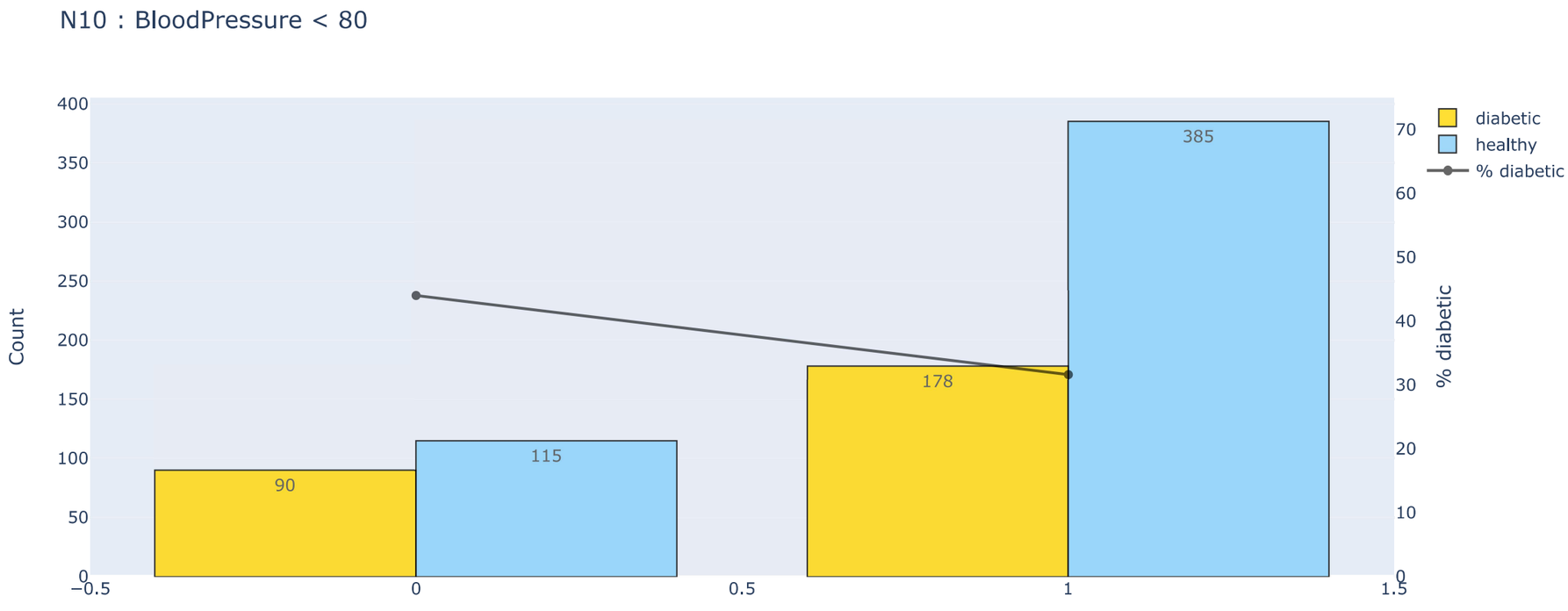
In [72]:
```
plot_pie('N9', 'Insulin < 200')
```



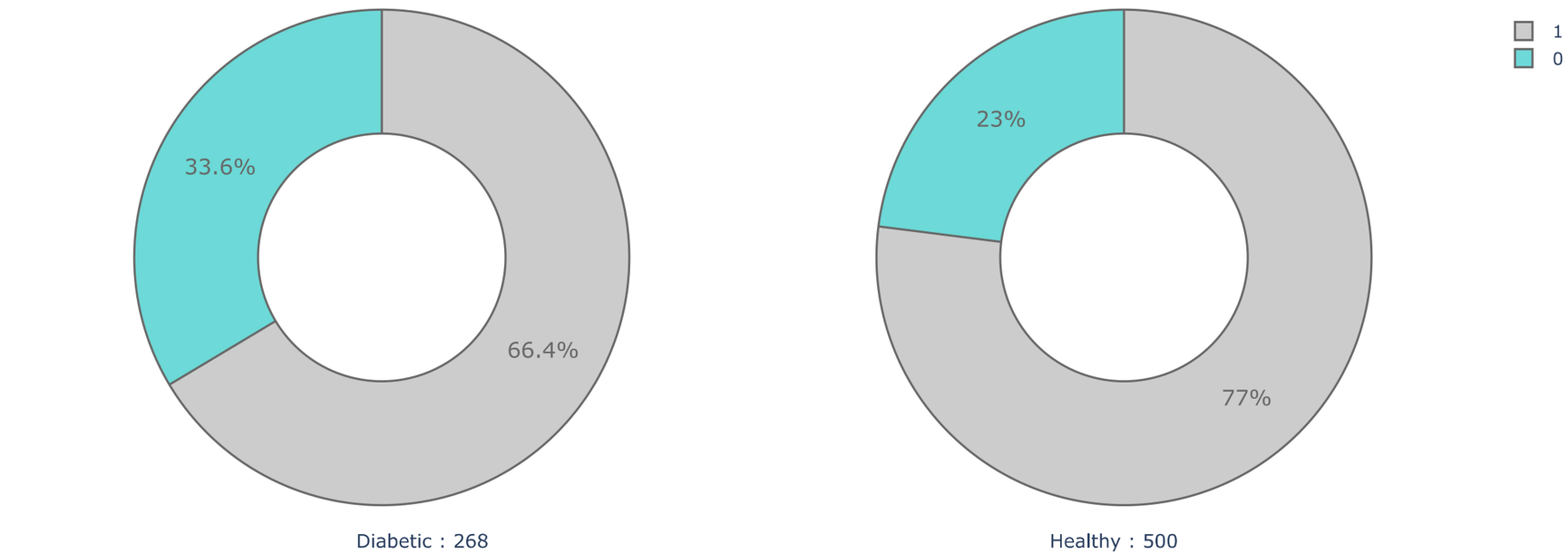
- **BloodPressure**

In [73]:
```
data.loc[:,'N10']=0
data.loc[(data['BloodPressure']<80),'N10']=1
```

In [74]:
```
barplot('N10', ': BloodPressure < 80')
```

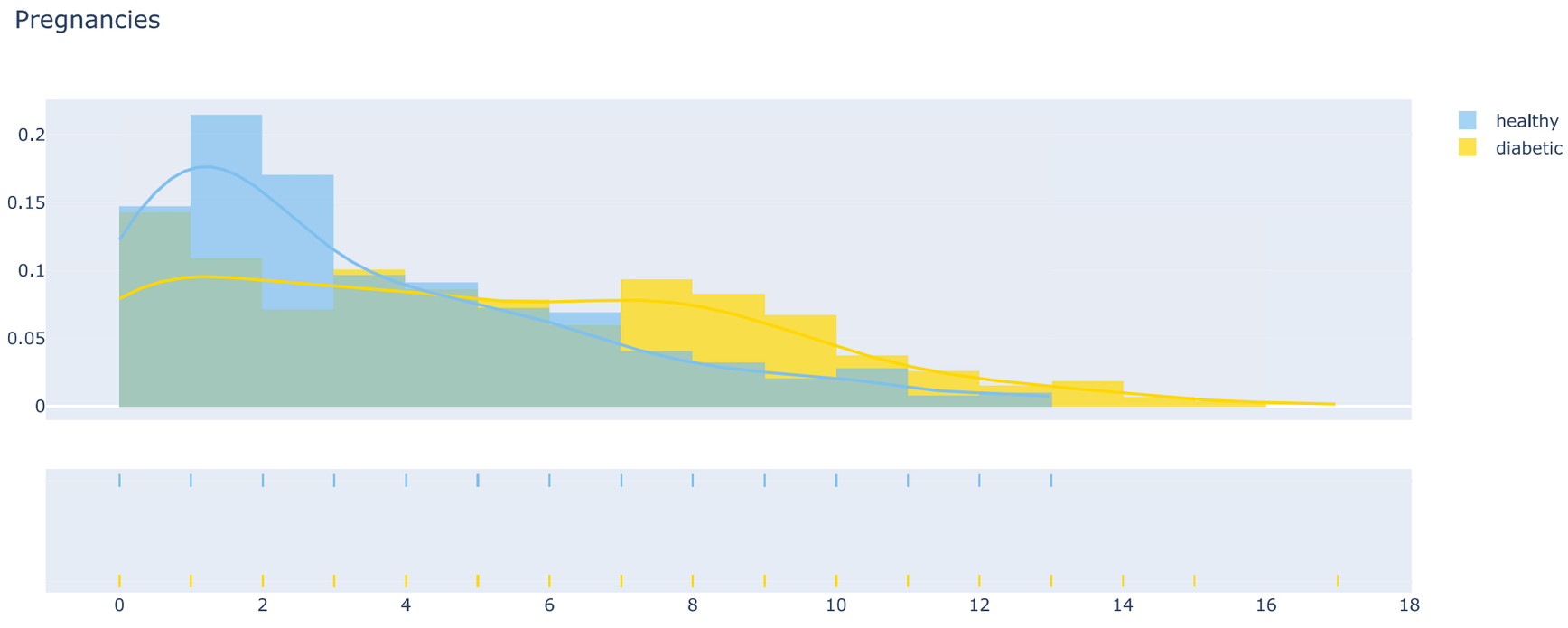


In [75]:
```
plot_pie('N10', 'BloodPressure < 80')
```



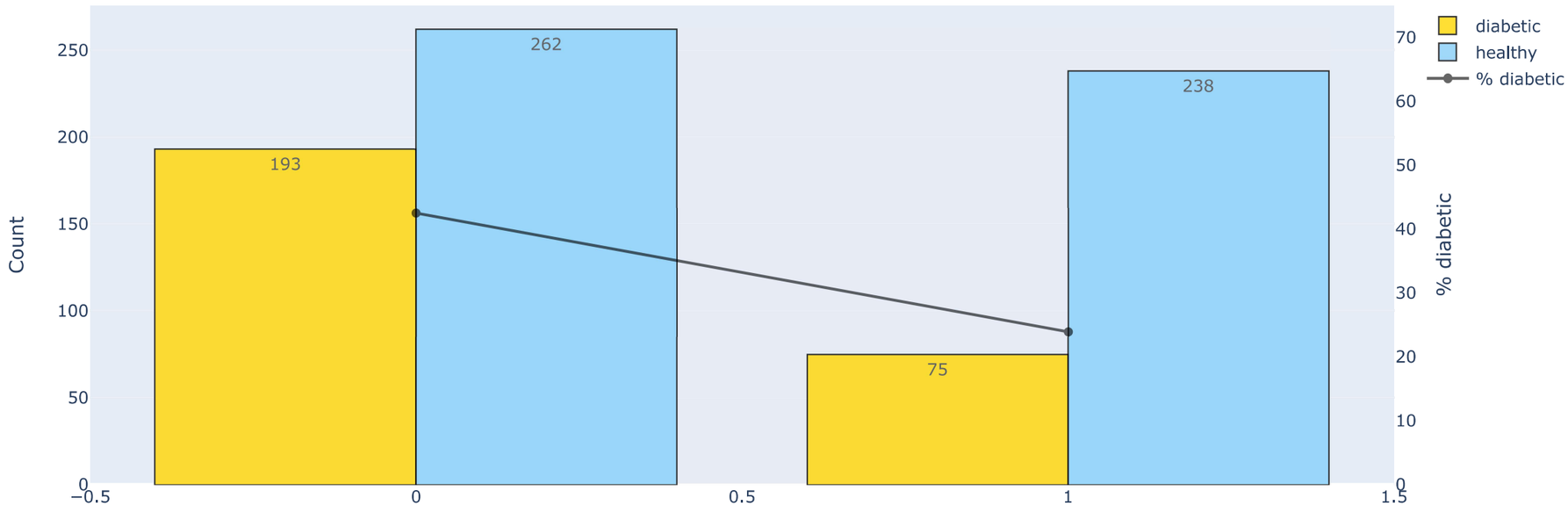
- **Pregnancies**

In [76]:
```
plot_distribution('Pregnancies', 0)
```

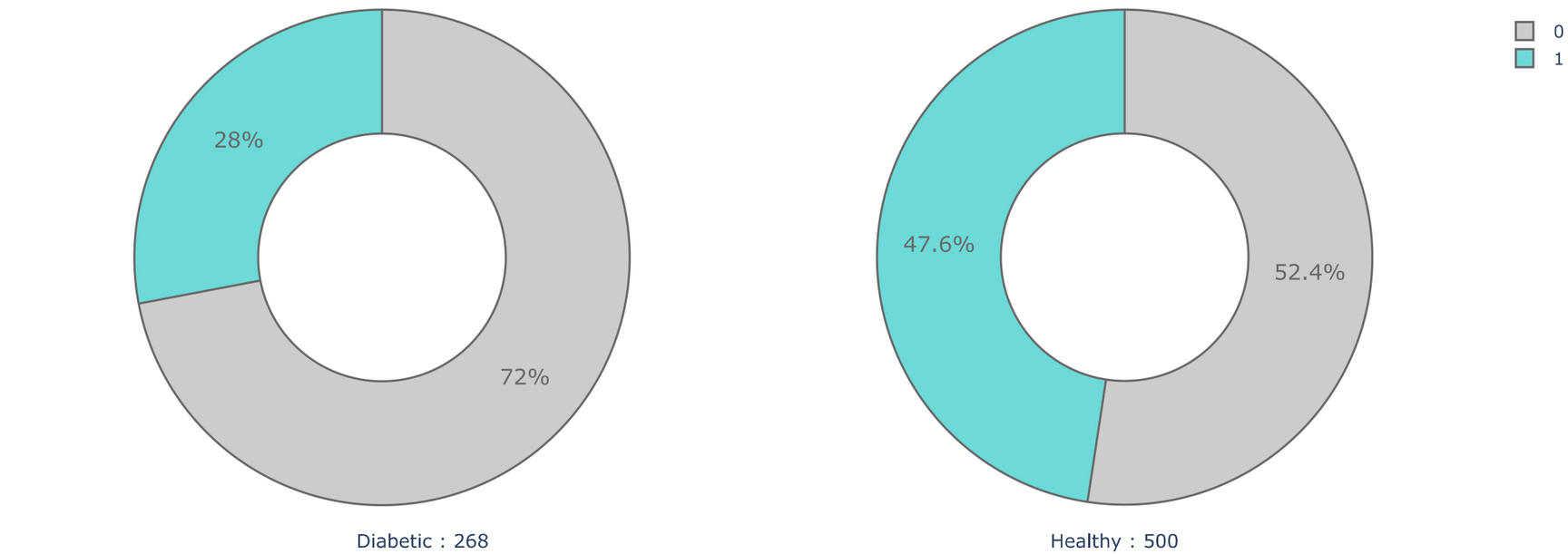


In [77]:
```
data.loc[:,'N11']=0
data.loc[(data['Pregnancies']<4) & (data['Pregnancies']!=0) ,'N11']=1
```

In [78]:
```
barplot('N11', ': Pregnancies > 0 and < 4')
```



In [79]:
```
plot_pie('N11', 'Pregnancies > 0 and < 4')
```

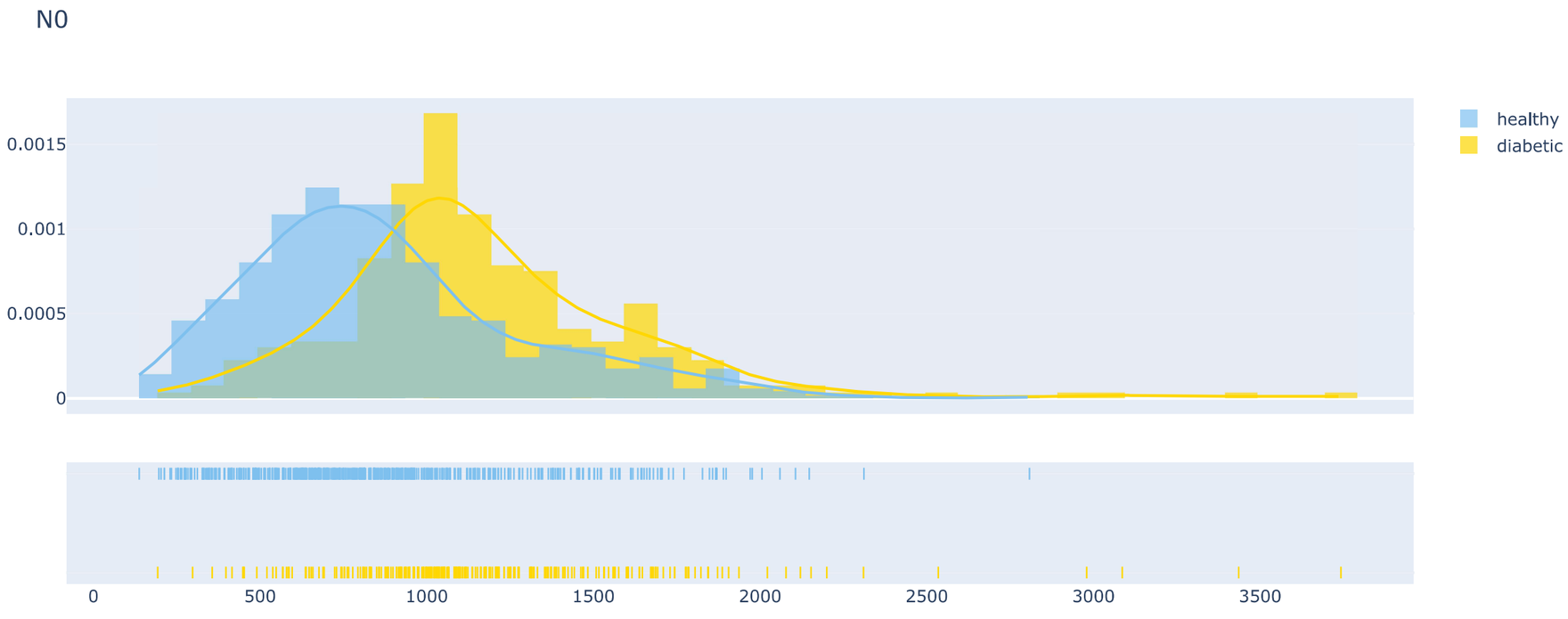


- **Others**

In [80]:
```
data['N0'] = data['BMI'] * data['SkinThickness']

data['N8'] =  data['Pregnancies'] / data['Age']

data['N13'] = data['Glucose'] / data['DiabetesPedigreeFunction']

data['N12'] = data['Age'] * data['DiabetesPedigreeFunction']

data['N14'] = data['Age'] / data['Insulin']
```

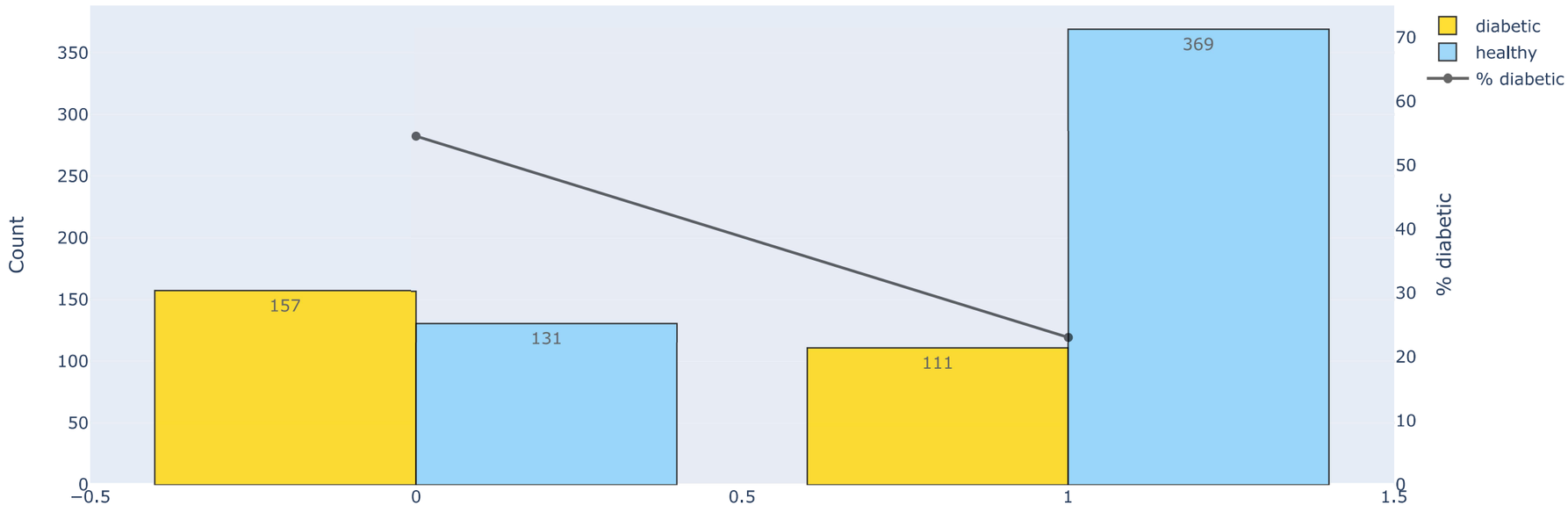
In [81]:
```
D = data[(data['Outcome'] != 0)]
H = data[(data['Outcome'] == 0)]
```

In [82]:
```
plot_distribution('N0', 0)
```

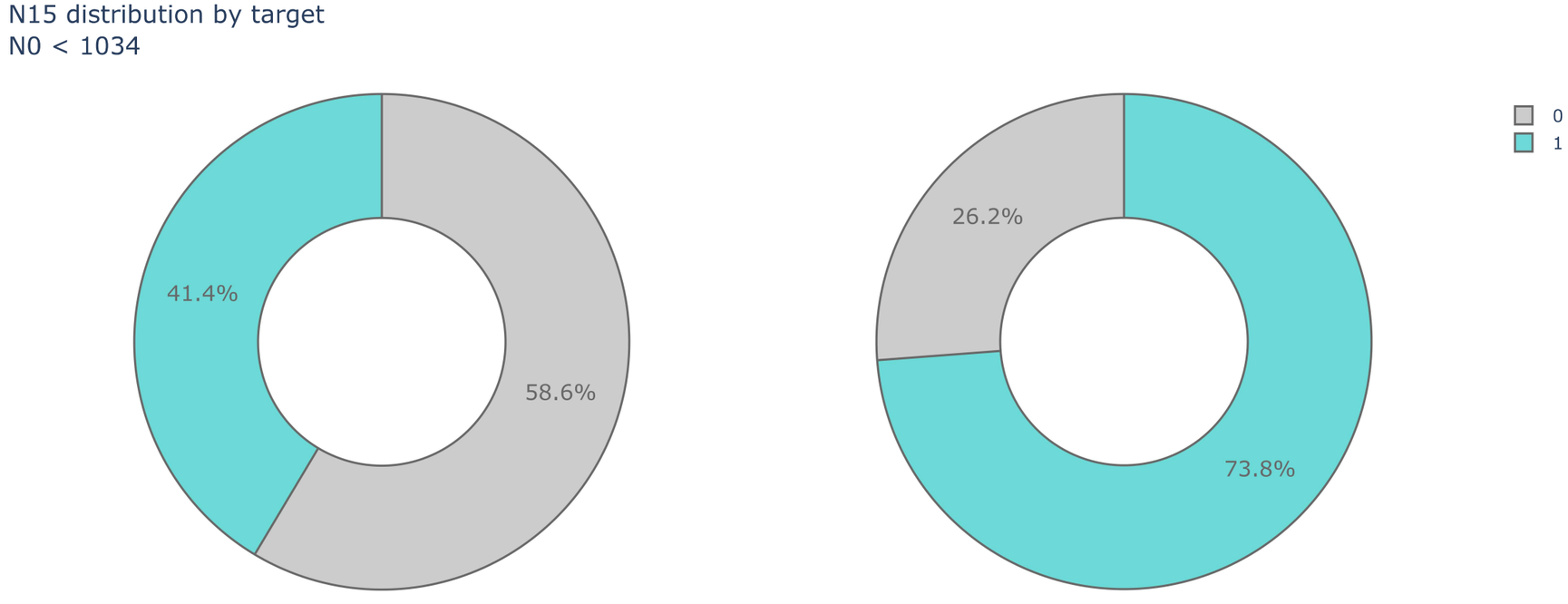


In [83]:
```
data.loc[:,'N15']=0
data.loc[(data['N0']<1034) ,'N15']=1
```

In [84]:
```
barplot('N15', ': N0 < 1034')
```



In [85]:
```
plot_pie('N15', 'N0 < 1034')
```



5. Prepare dataset

5.1. StandardScaler and LabelEncoder

- **StandardScaler** :

To standardize features, the process involves removing the mean and scaling them to unit variance. Centering and scaling occur independently for each feature by calculating relevant statistics from the samples in the dataset. The mean and standard deviation are then retained to be applied to later data using the transform method.

Standardizing a dataset is a common necessity for many machine learning estimators. If individual features do not closely resemble standard normally distributed data (e.g., Gaussian distribution with a mean of 0 and variance of 1), these estimators might perform poorly.

- **LabelEncoder** : Encode labels with value between 0 and n_classes-1.

Bellow we encode the data to feed properly to our algorithm

In [86]:
```
target_col = ["Outcome"]
cat_cols   = data.nunique()[data.nunique() < 12].keys().tolist()
cat_cols   = [x for x in cat_cols ]
```

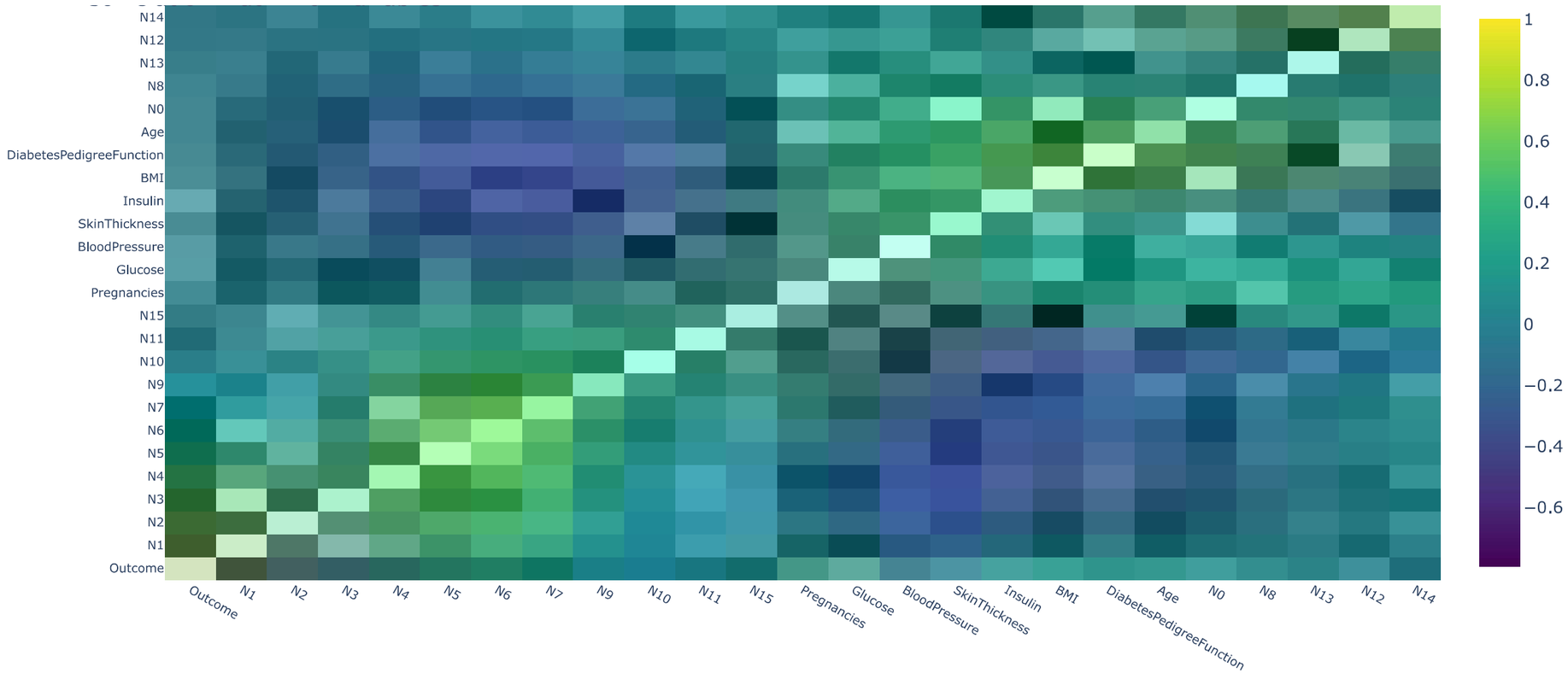
```
#numerical columns
num_cols   = [x for x in data.columns if x not in cat_cols + target_col]
#Binary columns with 2 values
bin_cols   = data.nunique()[data.nunique() == 2].keys().tolist()
#Columns more than 2 values
multi_cols = [i for i in cat_cols if i not in bin_cols]

#Label encoding Binary columns
le = LabelEncoder()
for i in bin_cols :
    data[i] = le.fit_transform(data[i])

#Duplicating columns for multi value columns
data = pd.get_dummies(data = data,columns = multi_cols )

#Scaling Numerical columns
std = StandardScaler()
scaled = std.fit_transform(data[num_cols])
scaled = pd.DataFrame(scaled,columns=num_cols)

#dropping original values merging scaled values for numerical columns
df_data_og = data.copy()
data = data.drop(columns = num_cols,axis = 1)
data = data.merge(scaled,left_index=True,right_index=True,how = "left")
```

5.2. Correlation Matrix

A correlation matrix is a table displaying correlation coefficients among sets of variables. Each random variable (Xi) in the table is correlated with every other value (Xj) in the same table. This visualization helps identify which pairs exhibit the highest correlation.

In [87]:
```
def correlation_plot():
    #correlation
    correlation = data.corr()
    #tick labels
    matrix_cols = correlation.columns.tolist()
    #convert to array
    corr_array  = np.array(correlation)
    trace = go.Heatmap(z = corr_array,
                       x = matrix_cols,
                       y = matrix_cols,
                       colorscale='Viridis',
                       colorbar   = dict() ,
                      )
    layout = go.Layout(dict(title = 'Correlation Matrix for variables',
                            #autosize = False,
                            #height  = 1400,
                            #width   = 1600,
                            margin  = dict(r = 0 ,l = 100,
                                           t = 0,b = 100,
                                          ),
                            yaxis   = dict(tickfont = dict(size = 9)),
                            xaxis   = dict(tickfont = dict(size = 9)),
                           )
                      )
    fig = go.Figure(data = [trace],layout = layout)
    py.iplot(fig)
```

In [88]:
```
correlation_plot()
```



5.3. X and y

In [89]:
```
# Def X and Y
X = data.drop('Outcome', 1)
y = data['Outcome']
```

5.4. Model Performance

To evaluate the performance of a model, several elements are crucial:

Confusion Matrix: Also known as the error matrix, it provides a visual representation of the algorithm's performance:

True Positive (TP): Diabetic cases correctly identified as diabetic

True Negative (TN): Healthy cases correctly identified as healthy

False Positive (FP): Healthy cases incorrectly identified as diabetic

False Negative (FN): Diabetic cases incorrectly identified as healthy

Metrics:

Accuracy: (TP + TN) / (TP + TN + FP + FN)

Precision: TP / (TP + FP)

Recall: TP / (TP + FN)

F1 Score: 2 *((Precision* Recall) / (Precision + Recall))

ROC Curve: This curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.

Precision-Recall Curve: It illustrates the tradeoff between precision and recall for different thresholds.

For training and testing machine learning algorithms, K-Fold cross-validation is utilized.

In K-fold cross-validation, the original sample is randomly divided into k equally-sized subsamples. Of these k subsamples, one subsample is retained as the validation data for testing the model, while the remaining k − 1 subsamples are employed as training data. This cross-validation process is then repeated k times, utilizing each of the k subsamples exactly once as the validation data. The results from the k iterations are then averaged to generate a single estimation. This method offers an advantage over repeated random sub-sampling, as it ensures that all observations are utilized for both training and validation, with each observation employed for validation exactly once.

Define a stylized report.

In [98]:
```
def model_performance(model, subtitle) :
    #Kfold
    cv = KFold(n_splits=5, shuffle=True, random_state=42)

    y_real = []
    y_proba = []
    tprs = []
    aucs = []
    mean_fpr = np.linspace(0,1,100)
    i = 1

    for train,test in cv.split(X,y):
        model.fit(X.iloc[train], y.iloc[train])
        pred_proba = model.predict_proba(X.iloc[test])
        precision, recall, _ = precision_recall_curve(y.iloc[test], pred_proba[:,1])
        y_real.append(y.iloc[test])
        y_proba.append(pred_proba[:,1])
        fpr, tpr, t = roc_curve(y[test], pred_proba[:, 1])
        tprs.append(interp(mean_fpr, fpr, tpr))
        roc_auc = auc(fpr, tpr)
        aucs.append(roc_auc)

    # Confusion matrix
    y_pred = cross_val_predict(model, X, y, cv=5)
    conf_matrix = confusion_matrix(y, y_pred)
    trace1 = go.Heatmap(z = conf_matrix  ,x = ["0 (pred)","1 (pred)"],
                        y = ["0 (true)","1 (true)"],xgap = 2, ygap = 2,
                        colorscale = 'Viridis', showscale  = False)

    #Show metrics
    tp = conf_matrix[1,1]
```

```
fn = conf_matrix[1,0]
fp = conf_matrix[0,1]
tn = conf_matrix[0,0]
Accuracy  =  ((tp+tn)/(tp+tn+fp+fn))
Precision =  (tp/(tp+fp))
Recall    =  (tp/(tp+fn))
F1_score  =  (2*(((tp/(tp+fp))*(tp/(tp+fn)))/((tp/(tp+fp))+(tp/(tp+fn)))))

show_metrics = pd.DataFrame(data=[[Accuracy , Precision, Recall, F1_score]])
show_metrics = show_metrics.T

colors = ['gold', 'lightgreen', 'lightcoral', 'lightskyblue']
trace2 = go.Bar(x = (show_metrics[0].values),
                y = ['Accuracy', 'Precision', 'Recall', 'F1_score'], text = np.round_(show_metrics[0].values,4),
                textposition = 'auto', textfont=dict(color='black'),
                orientation = 'h', opacity = 1, marker=dict(
        color=colors,
        line=dict(color='#000000',width=1.5)))

#Roc curve
mean_tpr = np.mean(tprs, axis=0)
mean_auc = auc(mean_fpr, mean_tpr)

trace3 = go.Scatter(x=mean_fpr, y=mean_tpr,
                    name = "Roc : " ,
                    line = dict(color = ('rgb(22, 96, 167)'),width = 2), fill='tozeroy')
trace4 = go.Scatter(x = [0,1],y = [0,1],
                    line = dict(color = ('black'),width = 1.5,
                    dash = 'dot'))

#Precision - recall curve
y_real = y
y_proba = np.concatenate(y_proba)
precision, recall, _ = precision_recall_curve(y_real, y_proba)

trace5 = go.Scatter(x = recall, y = precision,
                    name = "Precision" + str(precision),
                    line = dict(color = ('lightcoral'),width = 2), fill='tozeroy')

mean_auc=round(mean_auc,3)
#Subplots
fig = tls.make_subplots(rows=2, cols=2, print_grid=False,
                      specs=[[{}, {}],
                             [{}, {}]],
                      subplot_titles=('Confusion Matrix',
                                      'Metrics',
                                      'ROC curve'+" "+ '('+ str(mean_auc)+')',
                                      'Precision - Recall curve',
                                      ))
#Trace and layout
fig.append_trace(trace1,1,1)
fig.append_trace(trace2,1,2)
fig.append_trace(trace3,2,1)
fig.append_trace(trace4,2,1)
fig.append_trace(trace5,2,2)

fig['layout'].update(showlegend = False, title = '<b>Model performance report (5 folds)</b><br>'+subtitle,
                    autosize = False, height = 830, width = 830,
                    plot_bgcolor = 'black',
                    paper_bgcolor = 'black',
```

```
                      margin = dict(b = 195), font=dict(color='white'))
    fig["layout"]["xaxis1"].update(color = 'white')
    fig["layout"]["yaxis1"].update(color = 'white')
    fig["layout"]["xaxis2"].update((dict(range=[0, 1], color = 'white')))
    fig["layout"]["yaxis2"].update(color = 'white')
    fig["layout"]["xaxis3"].update(dict(title = "false positive rate"), color = 'white')
    fig["layout"]["yaxis3"].update(dict(title = "true positive rate"),color = 'white')
    fig["layout"]["xaxis4"].update(dict(title = "recall"), range = [0,1.05],color = 'white')
    fig["layout"]["yaxis4"].update(dict(title = "precision"), range = [0,1.05],color = 'white')
    for i in fig['layout']['annotations']:
        i['font'] = titlefont=dict(color='white', size = 14)
    py.iplot(fig)
```

5.5. Scores Tables

Complete model performance report with a table contain all results by fold

In [99]:
```
def scores_table(model, subtitle):
    scores = ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']
    res = []
    for sc in scores:
        scores = cross_val_score(model, X, y, cv = 5, scoring = sc)
        res.append(scores)
    df = pd.DataFrame(res).T
    df.loc['mean'] = df.mean()
    df.loc['std'] = df.std()
    df= df.rename(columns={0: 'accuracy', 1:'precision', 2:'recall',3:'f1',4:'roc_auc'})

    trace = go.Table(
        header=dict(values=['<b>Fold', '<b>Accuracy', '<b>Precision', '<b>Recall', '<b>F1 score', '<b>Roc auc'],
                    line = dict(color='#7D7F80'),
                    fill = dict(color='#a1c3d1'),
                    align = ['center'],
                    font = dict(size = 15)),
        cells=dict(values=[('1','2','3','4','5','mean', 'std'),
                           np.round(df['accuracy'],3),
                           np.round(df['precision'],3),
                           np.round(df['recall'],3),
                           np.round(df['f1'],3),
                           np.round(df['roc_auc'],3)],
                   line = dict(color='#7D7F80'),
                   fill = dict(color='#EDFAFF'),
                   align = ['center'], font = dict(size = 15)))

    layout = dict(width=800, height=400, title = '<b>Cross Validation - 5 folds</b><br>'+subtitle, font = dict(size = 15))
    fig = dict(data=[trace], layout=layout)

    py.iplot(fig, filename = 'styled_table')
```

6. Machine Learning

6.1. RandomSearch + LightGBM

LightGBM is a gradient boosting framework that utilizes tree-based learning algorithms. It is designed to be distributed and efficient, offering the following advantages:

Faster training speed and higher efficiency.

Lower memory usage.

Improved accuracy.

Support for parallel and GPU learning.

Capable of handling large-scale data.

To determine the optimal hyperparameters, we'll employ Random Search CV.

Random search is a technique that utilizes random combinations of hyperparameters to find the best solution for the model being constructed. Generally, Random Search is faster and more accurate compared to GridSearchCV, which computes all possible combinations. With Random Grid, we specify the number of combinations we wish to evaluate.

LightGBM: Hyperparameters

learning_rate: Determines the impact of each tree on the final outcome. It controls the magnitude of changes in the estimates during the updating process in the GBM.

n_estimators: Represents the number of trees or rounds in the model.

num_leaves: Specifies the number of leaves in the full tree; the default is 31.

min_child_samples: Indicates the minimum number of data points required in a single leaf. It can mitigate overfitting.

min_child_weight: Represents the minimum sum hessian in a single leaf.

subsample: Randomly selects a portion of data without resampling.

max_depth: Describes the maximum depth of the tree, essential for handling model overfitting.

colsample_bytree: LightGBM randomly selects a portion of features in each iteration if colsample_bytree is less than 1.0. For instance, setting it to 0.8 results in LightGBM selecting 80% of the features before training each tree.

reg_alpha: Refers to regularization techniques.

reg_lambda: Another parameter for regularization.

early_stopping_rounds: An option to expedite analysis. The model halts training if one validation dataset's metric fails to improve over the last 'early_stopping_round' rounds, reducing excessive iterations.

In [100…

```
random_state=42

fit_params = {"early_stopping_rounds" : 100,
              "eval_metric" : 'auc',
              "eval_set" : [(X,y)],
              'eval_names': ['valid'],
              'verbose': 0,
              'categorical_feature': 'auto'}

param_test = {'learning_rate' : [0.01, 0.02, 0.03, 0.04, 0.05, 0.08, 0.1, 0.2, 0.3, 0.4],
              'n_estimators' : [100, 200, 300, 400, 500, 600, 800, 1000, 1500, 2000],
              'num_leaves': sp_randint(6, 50),
              'min_child_samples': sp_randint(100, 500),
```

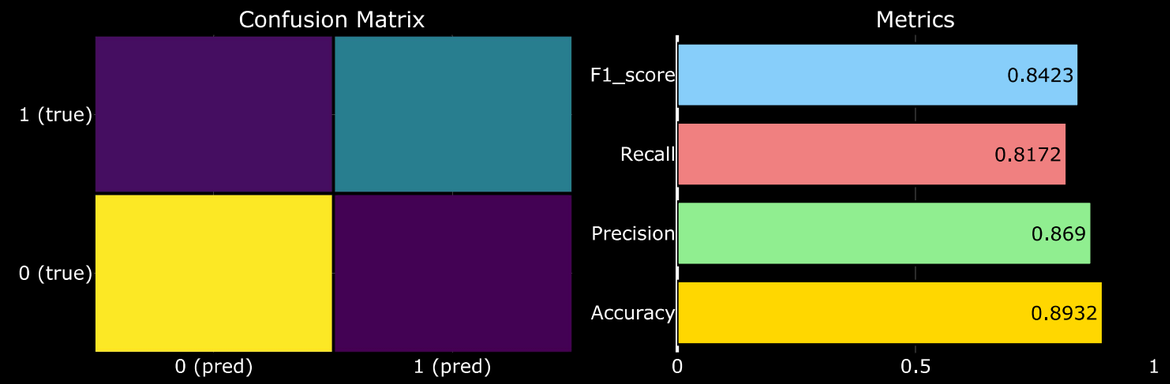
```
                'min_child_weight': [1e-5, 1e-3, 1e-2, 1e-1, 1, 1e1, 1e2, 1e3, 1e4],
                'subsample': sp_uniform(loc=0.2, scale=0.8),
                'max_depth': [-1, 1, 2, 3, 4, 5, 6, 7],
                'colsample_bytree': sp_uniform(loc=0.4, scale=0.6),
                'reg_alpha': [0, 1e-1, 1, 2, 5, 7, 10, 50, 100],
                'reg_lambda': [0, 1e-1, 1, 5, 10, 20, 50, 100]}

#number of combinations
n_iter = 300

#intialize lgbm and lunch the search
lgbm_clf = lgbm.LGBMClassifier(random_state=random_state, silent=True, metric='None', n_jobs=4)
grid_search = RandomizedSearchCV(
    estimator=lgbm_clf, param_distributions=param_test,
    n_iter=n_iter,
    scoring='accuracy',
    cv=5,
    refit=True,
    random_state=random_state,
    verbose=True)

grid_search.fit(X, y, **fit_params)
opt_parameters =  grid_search.best_params_
lgbm_clf = lgbm.LGBMClassifier(**opt_parameters)
```

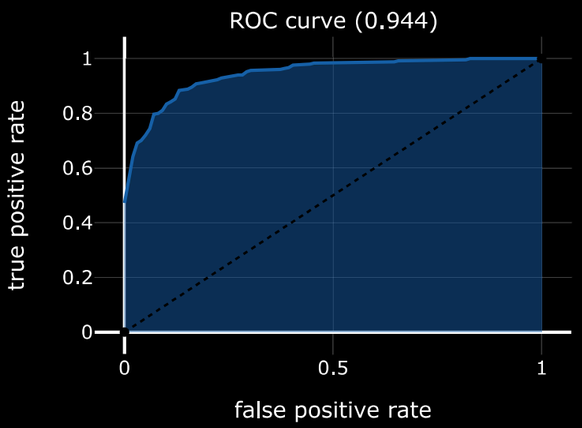
```
Fitting 5 folds for each of 300 candidates, totalling 1500 fits
```

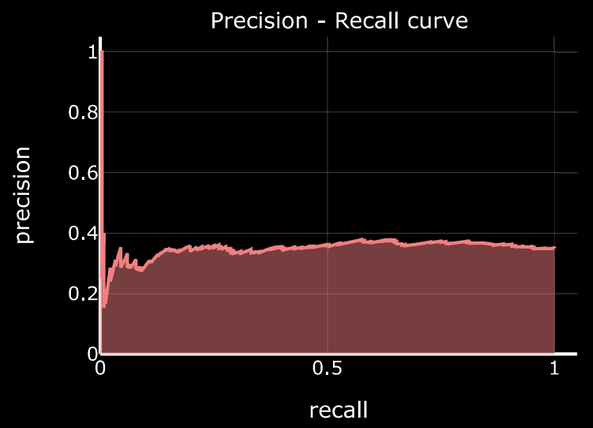
In [101…

```
model_performance(lgbm_clf, 'LightGBM')

scores_table(lgbm_clf, 'LightGBM')
```

Model performance report (5 folds)
LightGBM







Cross Validation - 5 folds

LightGBM

Fold	Accuracy	Precision	Recall	F1 score	Roc auc
1	0.89	0.878	0.796	0.835	0.943
2	0.864	0.811	0.796	0.804	0.929
3	0.877	0.857	0.778	0.816	0.934
4	0.908	0.898	0.83	0.863	0.963
5	0.928	0.904	0.887	0.895	0.972
mean	0.893	0.87	0.817	0.842	0.948
std	0.023	0.033	0.039	0.033	0.017

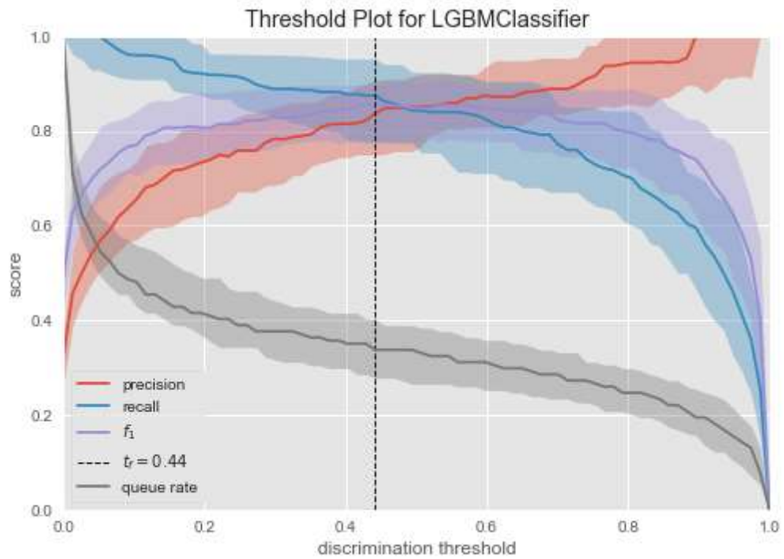
6.2. LightGBM - Discrimination Threshold

Discrimination Threshold:

This visualization represents precision, recall, F1 score, and queue rate concerning the discrimination threshold of a binary classifier. The discrimination threshold denotes the probability or score at which the positive class is chosen over the negative class.

In [102…

```
visualizer = DiscriminationThreshold(lgbm_clf)

visualizer.fit(X, y)
visualizer.poof()
```



Out[102]: <AxesSubplot:title={'center':'Threshold Plot for LGBMClassifier'}, xlabel='discrimination threshold', ylabel='score'>

6.3. GridSearch + LightGBM & KNN

We've achieved a significantly good result, but we can surpass 90% accuracy by incorporating a KNeighborsClassifier into LightGBM, using a Voting Classifier.

KNeighborsClassifier: This classifier operates by learning from the k-nearest neighbors of each query point, where the value of k is defined by the user.

VotingClassifier: It is a meta-classifier used to combine various machine learning classifiers, whether similar or conceptually different, for classification via majority or plurality voting.

With GridSearch CV we search the best "n_neighbors" to optimize accuracy of Voting Classifier

In [103…

```
knn_clf = KNeighborsClassifier()

voting_clf = VotingClassifier(estimators=[
    ('lgbm_clf', lgbm_clf),
    ('knn', KNeighborsClassifier())], voting='soft', weights = [1,1])

params = {
      'knn__n_neighbors': np.arange(1,30)
      }

grid = GridSearchCV(estimator=voting_clf, param_grid=params, cv=5)

grid.fit(X,y)

print("Best Score:" + str(grid.best_score_))
print("Best Parameters: " + str(grid.best_params_))
```

```
Best Score:0.8919701213818861
Best Parameters: {'knn__n_neighbors': 9}
```

With n_neighbors = 25, the accuracy increase to 90.625 ! Bellow the model performance report

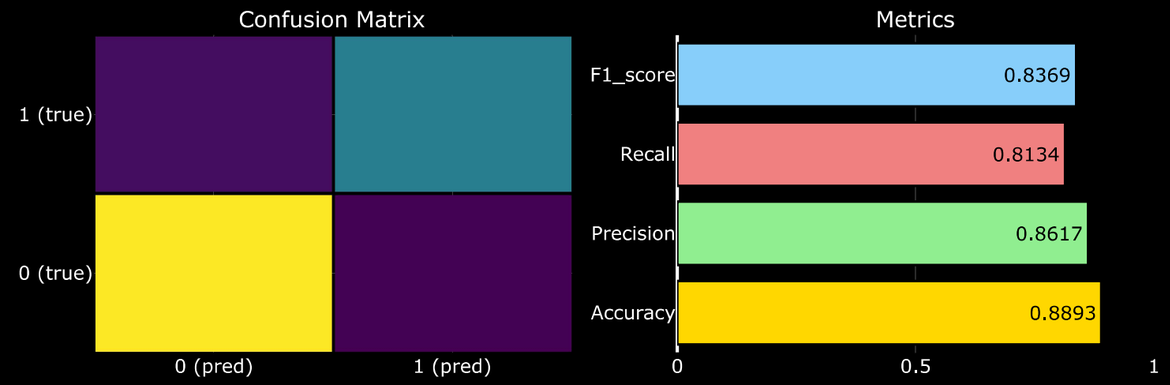
In [104…

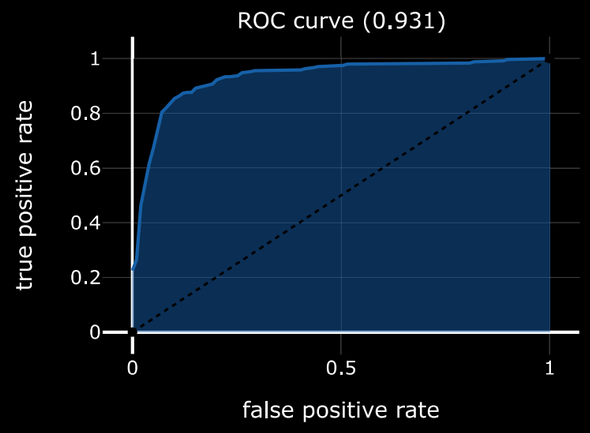
```
knn_clf = KNeighborsClassifier(n_neighbors = 25)

voting_clf = VotingClassifier (
        estimators = [('knn', knn_clf), ('lgbm', lgbm_clf)],
                      voting='soft', weights = [1,1])
```

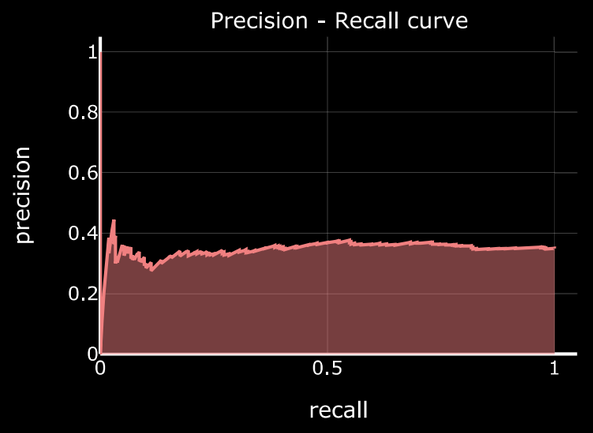
In [105…

```
model_performance(voting_clf, 'LightGBM & KNN')

scores_table(voting_clf, 'LightGBM & KNN')
```

Model performance report (5 folds)
LightGBM & KNN







Cross Validation - 5 folds

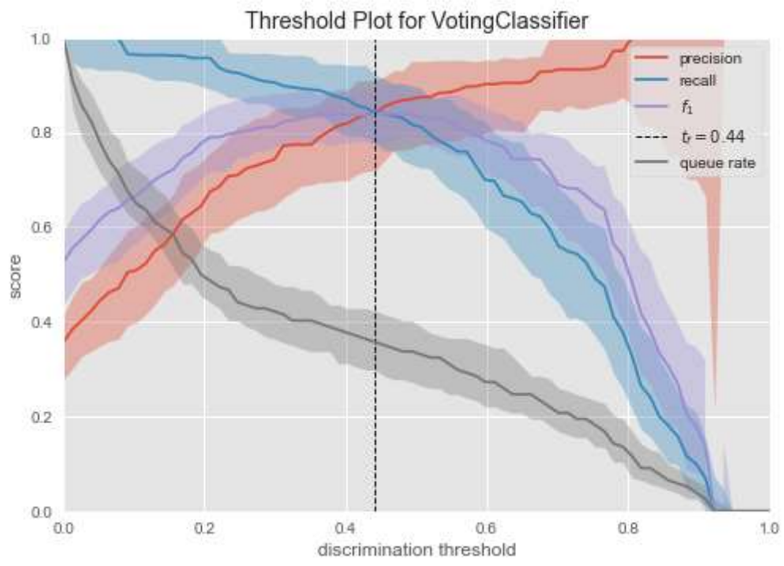
LightGBM & KNN

Fold	Accuracy	Precision	Recall	F1 score	Roc auc
1	0.903	0.915	0.796	0.851	0.92
2	0.864	0.8	0.815	0.807	0.924
3	0.87	0.854	0.759	0.804	0.928
4	0.908	0.898	0.83	0.863	0.956
5	0.902	0.852	0.868	0.86	0.957
mean	0.889	0.864	0.814	0.837	0.937
std	0.019	0.04	0.036	0.026	0.016

6.4. LightGBM & KNN - Discrimination Threshold

In [106…

```
visualizer = DiscriminationThreshold(voting_clf)

visualizer.fit(X, y)
visualizer.poof()
```



Out[106]:

```
<AxesSubplot:title={'center':'Threshold Plot for VotingClassifier'}, xlabel='discrimination threshold', ylabel='score'>
```