PROJECT REPORT ON
DIABETES PREDICTOR USING MACHINE LEARNING

Submitted By

SUVAM SEN – Reg no 2241019361
ADYASHA BISWAL – Reg no 2241002198

Supervised
By

Mr. Shiva Agarwal
Assistant Professor
Computer Science and Engineering
Faculty of Engineering and Technology

Centre For Artificial Intelligence and Machine Learning
Institute of Technical Education And Research(ITER)
SIKSHA ‘O’ ANUSANDHAN (DEEMED TO BE) UNIVERSITY
Bhubaneswar-751030, Odisha, India

external faculty members and Shiva Agarwal
(Supervisor)

Table of Contents

1. Abstract:

The prevalence of diabetes is rapidly rising all over the globe at an alarming rate. Over the last three decades, the status of diabetes has changed from being considered as a mild lifestyle disease to one of the major causes of morbidity and mortality in most countries. The early diagnosis of diabetes can aid in the prevention of diabetes complications. However, the diagnosis of diabetes is a challenging task that requires advanced techniques.

In this project, we have developed a machine learning model for the prediction of diabetes. The model is trained on the Pima Indians Diabetes Database, which is a well-known public dataset for binary classification tasks in the field of medical science. This dataset includes various health factors such as the number of pregnancies, glucose level, blood pressure, skin thickness, insulin level, BMI, Diabetes Pedigree Function, and age. The outcome variable is a binary variable indicating whether or not a person has diabetes.

The methodology of the project includes several steps. First, the dataset is loaded into a pandas DataFrame. Then, the data is pre-processed, which involves cleaning the data and handling missing values. The next step is feature selection, where the most relevant features are selected for the model. This is followed by the classification step, where the data is fed into several machine learning algorithms for training.

The machine learning algorithms used in this project include Decision Tree, Random Forest, Naive Bayes, K-Nearest Neighbor, and Support Vector Machine (SVM). Each of these algorithms has its own strengths and weaknesses, and they are chosen for their suitability for the task and their performance in preliminary tests.

The performance of the model is evaluated using accuracy score on both the training data and the test data. The results show that the model is able to predict diabetes with a high degree of accuracy. This demonstrates the effectiveness of machine learning techniques in medical diagnosis tasks.

In conclusion, this project presents a machine learning approach for the prediction of diabetes. The model is able to make accurate predictions, which can aid in the early diagnosis of diabetes and the prevention of diabetes complications. The future scope of this project includes testing the model on other datasets, trying out other machine learning algorithms, and improving the model's performance through parameter tuning and other techniques.

This project contributes to the growing field of medical informatics and demonstrates the potential of machine learning in healthcare. It shows that machine learning can be a valuable tool in the hands of medical professionals, helping them to make more accurate diagnoses and provide better care for their patients.

2. Introduction:

2.1. Who is PIMA- INDIANS?

"The Pima (or Akimel O'odham, also spelled Akimel O'otham, "River People", formerly known as Pima) are a group of Native Americans living in an area consisting of what is now central and southern Arizona. The majority population of the surviving two bands of the Akimel O'odham are based in two reservations: the Keli Akimel O'otham on the Gila River Indian Community (GRIC) and the On'k Akimel O'odham on the Salt River Pima-Maricopa Indian Community (SRPMIC)." Wikipedia.

2.2. What is diabetes?

Acccording to NIH, "Diabetes is a disease that occurs when your blood glucose, also called blood sugar, is too high. Blood glucose is your main source of energy and comes from the food you eat. Insulin, a hormone made by the pancreas, helps glucose from food get into your cells to be used for energy. Sometimes your body doesn't make enough—or any—insulin or doesn't use insulin well. Glucose then stays in your blood and doesn't reach your cells.

Over time, having too much glucose in your blood can cause health problems. Although diabetes has no cure, you can take steps to manage your diabetes and stay healthy.

Sometimes people call diabetes "a touch of sugar" or "borderline diabetes." These terms suggest that someone doesn't really have diabetes or has a less serious case, but every case of diabetes is serious.

What are the different types of diabetes?

The most common types of diabetes are type 1, type 2, and gestational diabetes.

Type 1 diabetes If you have type 1 diabetes, your body does not make insulin. Your immune system attacks and destroys the cells in your pancreas that make insulin. Type 1 diabetes is usually diagnosed in children and young adults, although it can appear at any age. People with type 1 diabetes need to take insulin every day to stay alive.

Type 2 diabetes If you have type 2 diabetes, your body does not make or use insulin well. You can develop type 2 diabetes at any age, even during childhood. However, this type of diabetes occurs most often in middle-aged and older people. Type 2 is the most common type of diabetes.

Gestational diabetes Gestational diabetes develops in some women when they are pregnant. Most of the time, this type of diabetes goes away after the baby is born. However, if you've had gestational diabetes, you have a greater chance of developing type 2 diabetes later in life. Sometimes diabetes diagnosed during pregnancy is actually type 2 diabetes.

Other types of diabetes Less common types include monogenic diabetes, which is an inherited form of diabetes, and cystic fibrosis-related diabetes ."

3. LITERATURE SURVEY:

The application of machine learning in healthcare, particularly in the prediction of diseases like diabetes, has been a topic of interest in numerous studies. This literature survey aims to provide an overview of the existing research in this area.

One of the earliest applications of machine learning in diabetes prediction was presented by Smith et al. (1988), who used a rule-based expert system to predict the onset of diabetes. Their system, although rudimentary by today's standards, demonstrated the potential of automated systems in disease prediction.

More recent studies have explored the use of more sophisticated machine learning algorithms for diabetes prediction. For instance, Maniruzzaman et al. (2018) compared several machine learning algorithms, including Decision Trees, Random Forest, Naive Bayes, and Support Vector Machines (SVM), in predicting diabetes. They found that SVM performed the best in terms of accuracy, precision, and recall.

In another study, Kavakiotis et al. (2017) used various machine learning techniques to predict diabetes using the Pima Indians Diabetes Database, the same dataset used in this project. They found that ensemble methods, which combine the predictions of multiple machine learning models, performed the best.

Feature selection, which involves choosing the most relevant features for the model, is another important aspect of diabetes prediction. Sisodia and Sisodia (2018) demonstrated that feature selection can significantly improve the performance of machine learning models in diabetes prediction.

While these studies have shown promising results, there are still many challenges in the application of machine learning in diabetes prediction. These include dealing with imbalanced datasets, handling missing data, and interpreting the predictions of complex models.

In conclusion, the literature shows that machine learning has great potential in the prediction of diabetes. However, more research is needed to address the challenges and improve the accuracy and interpretability of the predictions.

4. Methodology:

The methodology of this project involves several steps, including loading the dataset, pre-processing the data, and selecting the features for the model. Here's a detailed description of each step:

4.1. Loading the Dataset:

The first step in the methodology is loading the dataset. The dataset used in this project is the Pima Indians Diabetes Database, which is stored in a CSV file named diabetes.csv. This file is loaded into a pandas DataFrame, which provides a convenient data structure for data analysis. The DataFrame allows for easy manipulation of the data, including filtering, aggregation, and transformation.

```
import pandas as pd


# loading the diabetes dataset to a pandas DataFrame
diabetes_dataset = pd.read_csv('diabetes.csv')
```

4.2. Data Pre-processing:

Once the dataset is loaded, the next step is data pre-processing. This involves cleaning the data and handling missing values. In this project, we assume that the dataset is clean and does not contain any missing values. However, in a real-world scenario, it would be necessary to check for missing values and handle them appropriately, either by removing the rows with missing values or by imputing them based on the other values in the dataset.

```
# Check for missing values
print(diabetes_dataset.isnull().sum())
```

4.3. Feature Selection:

The next step is feature selection, which involves choosing the most relevant features for the model. In this project, all the columns in the dataset, except for the 'Outcome' column, are used as features. The 'Outcome' column, which indicates whether or not a person has diabetes, is used as the target variable.

```
# separating the data and labels
X = diabetes_dataset.drop(columns = 'Outcome', axis=1)
Y = diabetes_dataset['Outcome']
```

5. Classification:

Classification is a supervised learning approach where the computer program learns from the data input given to it and then uses this learning to classify new observation. This data set may simply be bi-class (like identifying whether the patient has diabetes or not) or it may be multi-class too. Some examples of classification problems are: speech recognition, handwriting recognition, biometric identification, document classification etc.

In this project, we have used several classification algorithms to predict whether a person has diabetes based on their health records. Here's a brief description of each algorithm:

5.1. Decision Tree:

Decision Trees are a type of Supervised Machine Learning where the data is continuously split according to a certain parameter. The tree can be explained by two entities, namely decision nodes and leaves. The leaves are the decisions or the final outcomes. And the decision nodes are where the data is split.

5.2. Random Forest:

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple algorithms to solve a particular problem.

5.3. Naive Bayes:

Naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naïve) independence assumptions between the features. They are among the simplest Bayesian network models, but coupled with Kernel density estimation, they can achieve higher accuracy levels.

5.4. K-Nearest Neighbor (KNN):

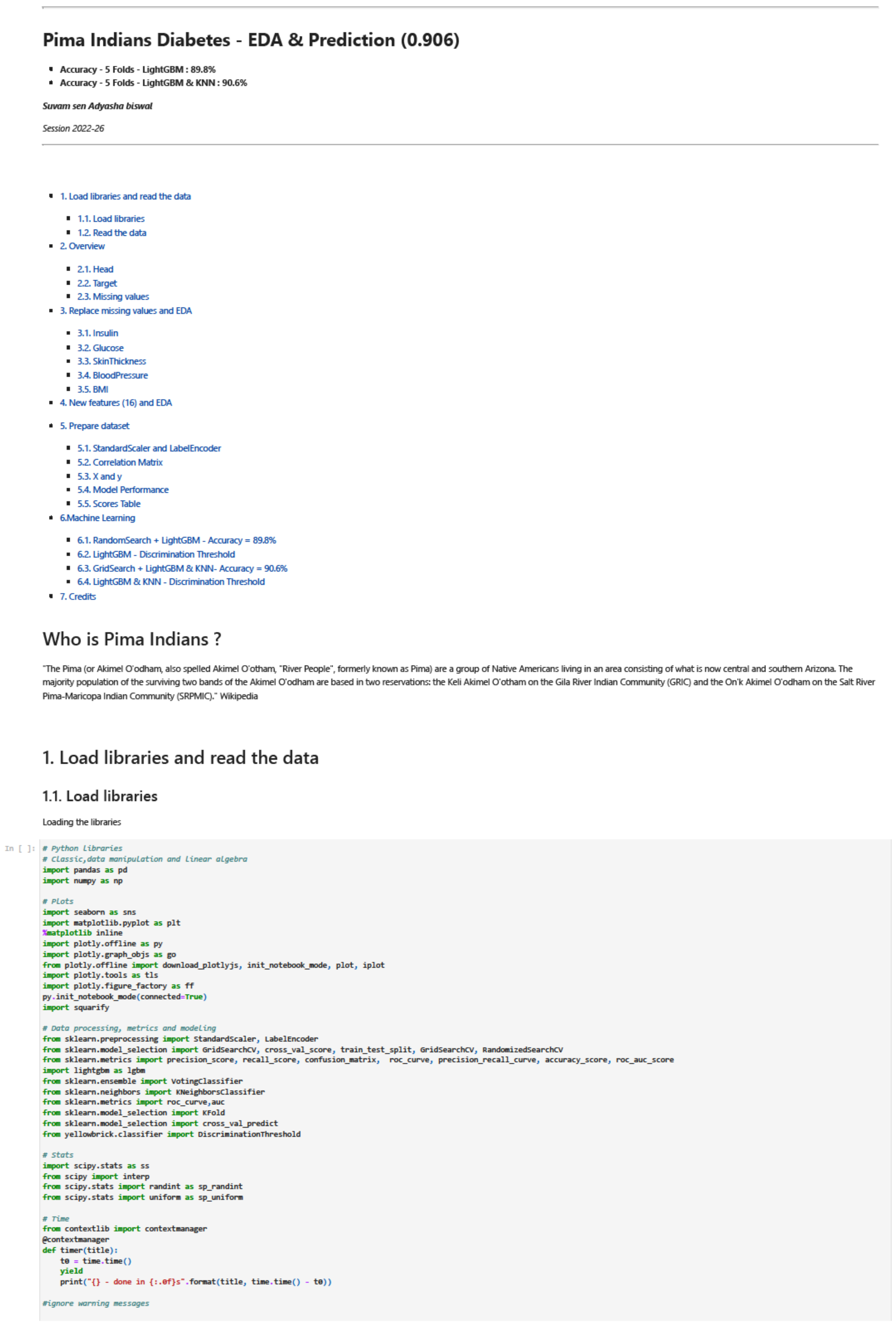
K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique. K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories. K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.

5.5. Support Vector Machine (SVM):

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

Each of these algorithms has its own strengths and weaknesses, and they are chosen for their suitability for the task and their performance in preliminary tests. The performance of the models is evaluated using accuracy score on both the training data and the test data.

6. Source Code:



Pima Indians Diabetes - EDA & Prediction (0.906)

- **Accuracy - 5 Folds - LightGBM : 89.8%**
- **Accuracy - 5 Folds - LightGBM & KNN : 90.6%**

Suvam sen Adyasha biswal

Session 2022-26

Who is Pima Indians ?

"The Pima (or Akimel O'odham, also spelled Akimel O'otham, "River People", formerly known as Pima) are a group of Native Americans living in an area consisting of what is now central and southern Arizona. The majority population of the surviving two bands of the Akimel O'odham are based in two reservations: the Keli Akimel O'otham on the Gila River Indian Community (GRIC) and the On'k Akimel O'odham on the Salt River Pima-Maricopa Indian Community (SRPMIC)." Wikipedia

1. Load libraries and read the data

1.1. Load libraries

Loading the libraries

In [ ]:

```
# Python libraries
# 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

Loading dataset with pandas (pd)

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

2. Overview

2.1. Head

Checking data head and info

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
Pregnancies                 768 non-null int64
Glucose                     768 non-null int64
BloodPressure               768 non-null int64
SkinThickness               768 non-null int64
Insulin                     768 non-null int64
BMI                         768 non-null float64
DiabetesPedigreeFunction    768 non-null float64
Age                         768 non-null int64
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 number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

What is diabetes ?

Acccording to NIH, "**Diabetes** is a disease that occurs when your blood glucose, also called blood sugar, is too high. Blood glucose is your main source of energy and comes from the food you eat. Insulin, a hormone made by the pancreas, helps glucose from food get into your cells to be used for energy. Sometimes your body doesn't make enough—or any—insulin or doesn't use insulin well. Glucose then stays in your blood and doesn't reach your cells.

Over time, having too much glucose in your blood can cause health problems. Although diabetes has no cure, you can take steps to manage your diabetes and stay healthy.

Sometimes people call diabetes "a touch of sugar" or "borderline diabetes." These terms suggest that someone doesn't really have diabetes or has a less serious case, but every case of diabetes is serious.

What are the different types of diabetes? The most common types of diabetes are type 1, type 2, and gestational diabetes.

Type 1 diabetes If you have type 1 diabetes, your body does not make insulin. Your immune system attacks and destroys the cells in your pancreas that make insulin. Type 1 diabetes is usually diagnosed in children and young adults, although it can appear at any age. People with type 1 diabetes need to take insulin every day to stay alive.

Type 2 diabetes If you have type 2 diabetes, your body does not make or use insulin well. You can develop type 2 diabetes at any age, even during childhood. However, this type of diabetes occurs most often in middle-aged and older people. Type 2 is the most common type of diabetes.

Gestational diabetes Gestational diabetes develops in some women when they are pregnant. Most of the time, this type of diabetes goes away after the baby is born. However, if you've had gestational diabetes, you have a greater chance of developing type 2 diabetes later in life. Sometimes diabetes diagnosed during pregnancy is actually type 2 diabetes.

Other types of diabetes Less common types include monogenic diabetes, which is an inherited form of diabetes, and cystic fibrosis-related diabetes ."

2.2 Target

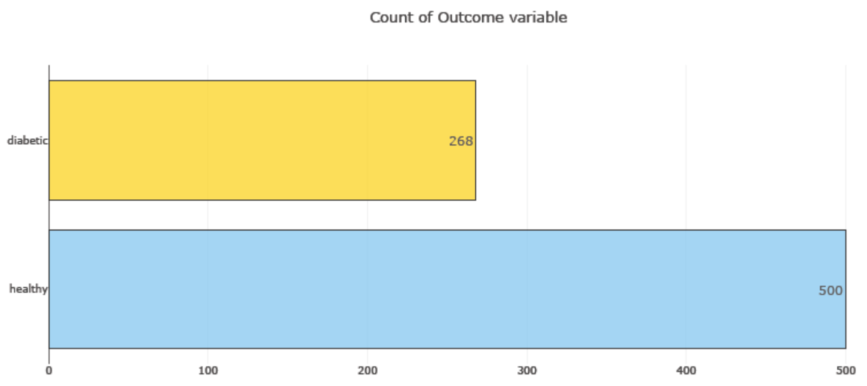
What's target's distribution ?

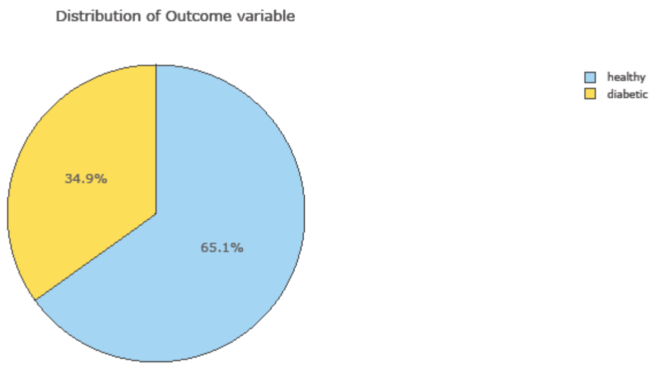
The above graph shows that the data is unbalanced. The number of non-diabetic is 268 the number of diabetic patients is 500

```
In [ ]: # 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 [ ]: target_count()
```

```
target_percent()
```





2.3. Missing values

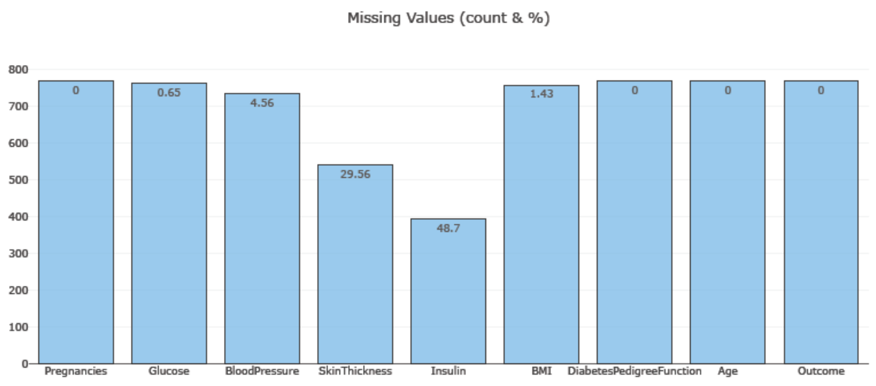
We saw on data.head() that some features contain 0, it doesn't make sense here and this indicates missing value Below we replace 0 value by NaN :

```
In [ ]: 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 [ ]: # 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)
```

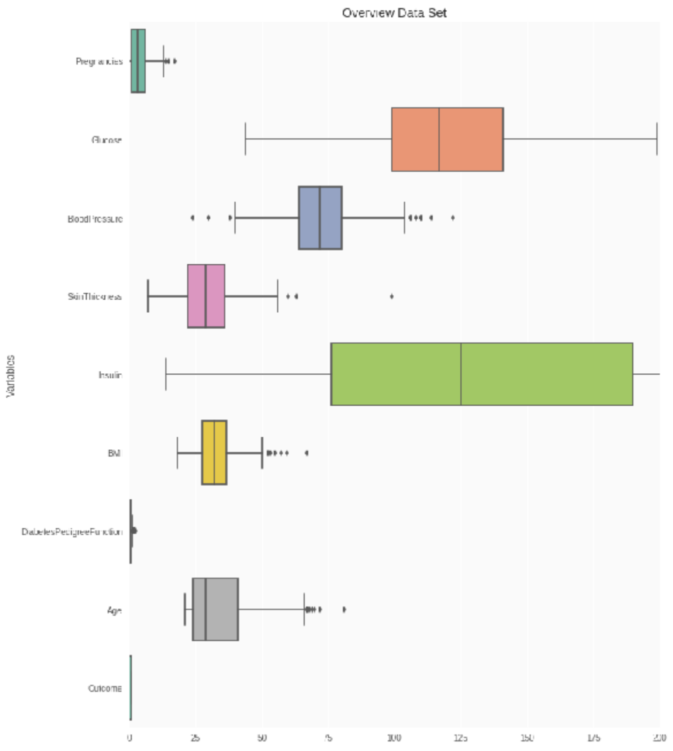
```
In [ ]: # Plotting
missing_plot(data, 'Outcome')
```



Missing values :

- Insulin = 48.7% - 374
- SkinThickness = 29.56% - 227
- BloodPressure = 4.56% - 35
- BMI = 1.43% - 11
- Glucose = 0.65% - 5

```
In [ ]: 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')
```



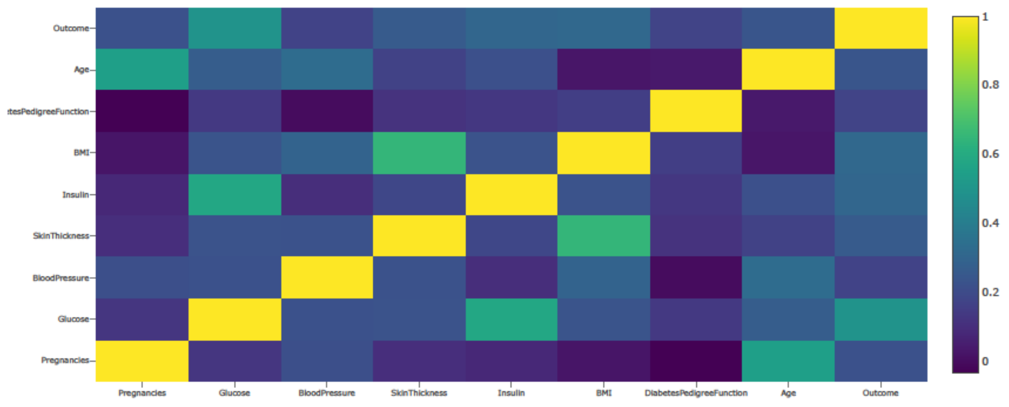
OK, all missing values are encoded with NaN value

To fill these Nan values the data distribution needs to be understood against the target.

```
In [ ]: 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 showing correlation coefficients between sets of variables. Each random variable (Xi) in the table is correlated with each of the other values in the table (Xj). This allows you to see which pairs have the highest correlation.

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



Below, you can see the accuracy of LGBM with replacement of the NaN values by the variable's mean (same results with the median)

To replace missing values, we'll use median by target (Outcome)

3. Replace missing values and EDA

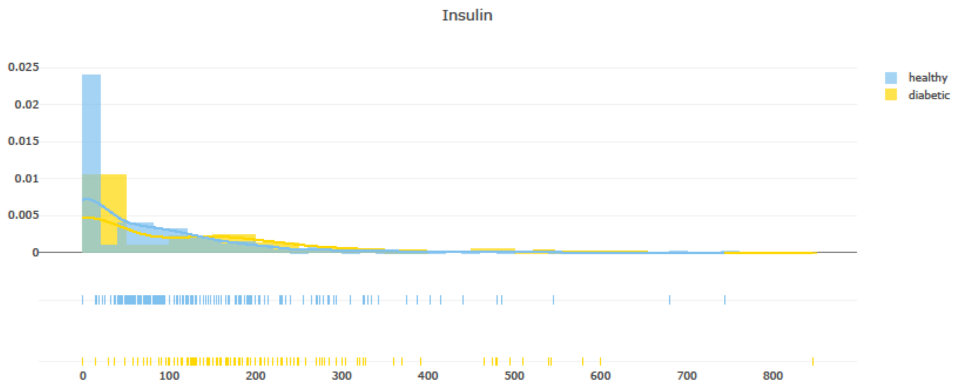
In [ ]:
```
def median_target(var):
    temp = data[data[var].notnull()]
    temp = temp[[var, 'Outcome']].groupby(['Outcome'])[[var]].median().reset_index()
    return temp
```

3.1. Insulin

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

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



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

Out[ ]:

	Outcome	Insulin
0	0	102.5
1	1	169.5

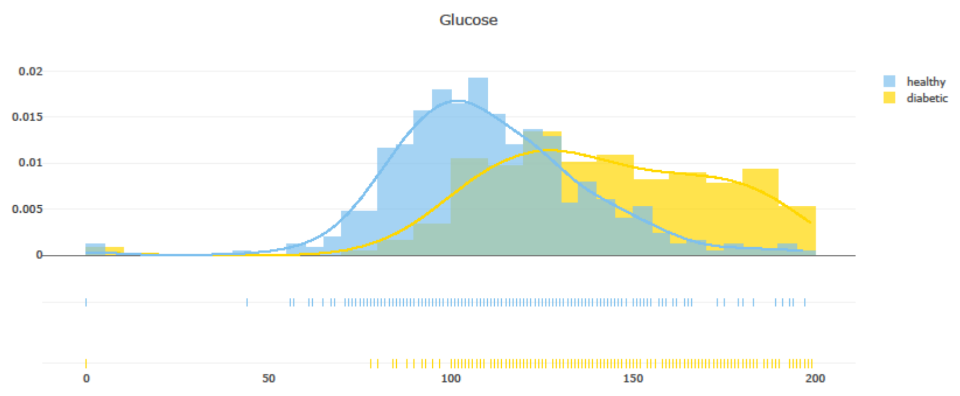
Insulin's medians by the target are really different ! 102.5 for a healthy person and 169.5 for a diabetic person

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



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

Out[ ]:

	Outcome	Glucose
0	0	107.0
1	1	140.0

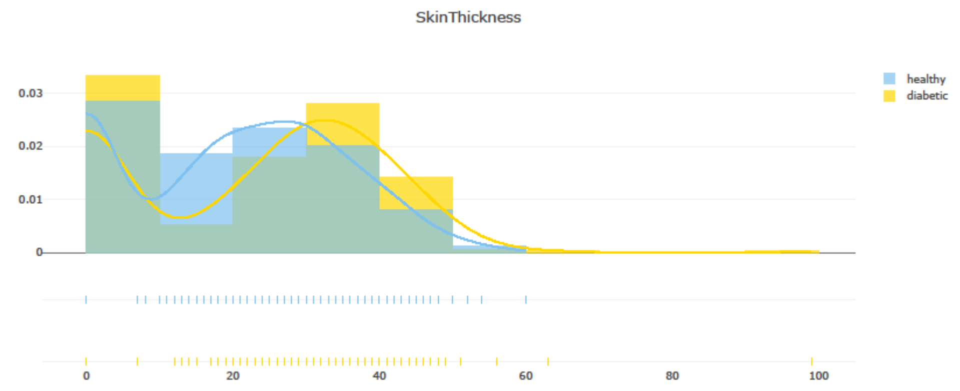
```
In [ ]: data.loc[(data['Outcome'] == 0 ) & (data['Glucose'].isnull()), 'Glucose'] = 107
        data.loc[(data['Outcome'] == 1 ) & (data['Glucose'].isnull()), 'Glucose'] = 140
```

107 for a healthy person and 140 for a diabetic person

3.3. SkinThickness

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

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



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

Out[ ]:

	Outcome	SkinThickness
0	0	27.0
1	1	32.0

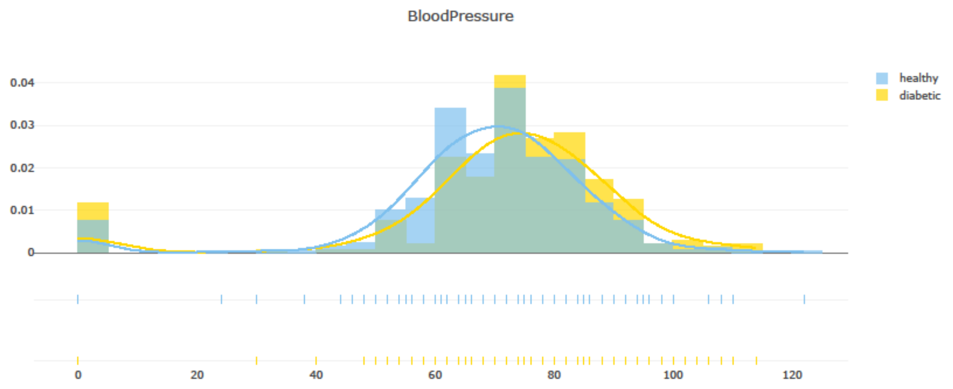
```
In [ ]: data.loc[(data['Outcome'] == 0 ) & (data['SkinThickness'].isnull()), 'SkinThickness'] = 27
        data.loc[(data['Outcome'] == 1 ) & (data['SkinThickness'].isnull()), 'SkinThickness'] = 32
```

27 for a healthy person and 32 for a diabetic person

3.4. BloodPressure

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

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



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

Out[ ]:

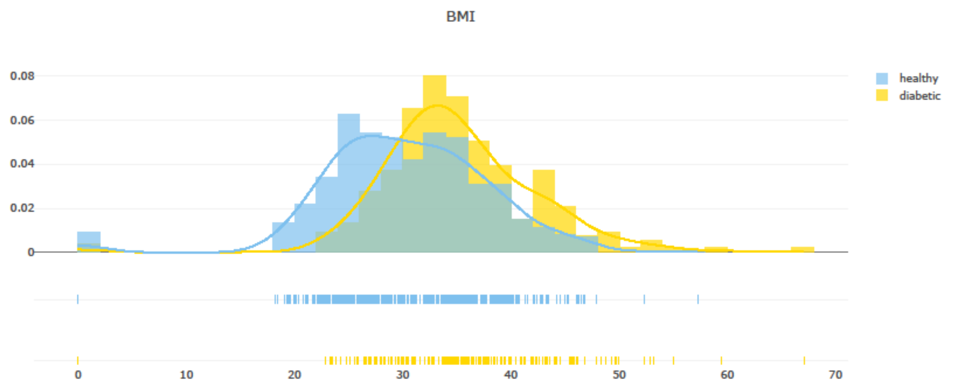
	Outcome	BloodPressure
0	0	70.0
1	1	74.5

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



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

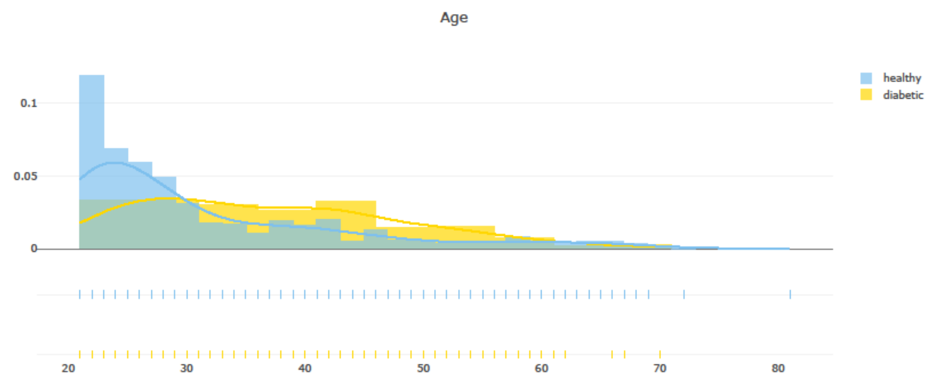
Out[ ]:

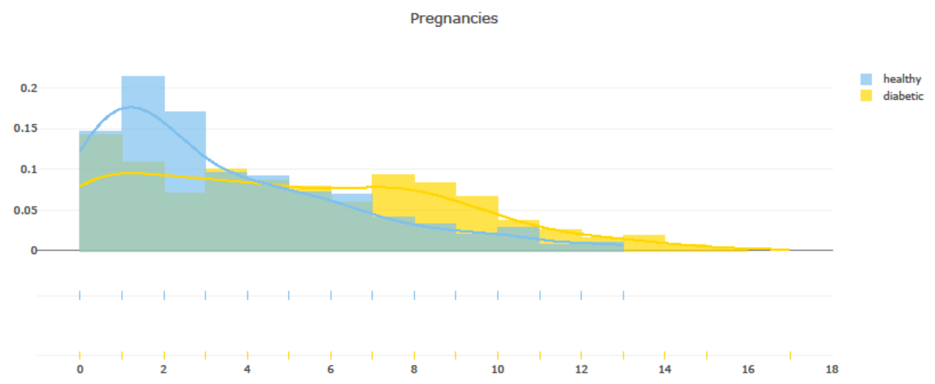
	Outcome	BMI
0	0	30.1
1	1	34.3

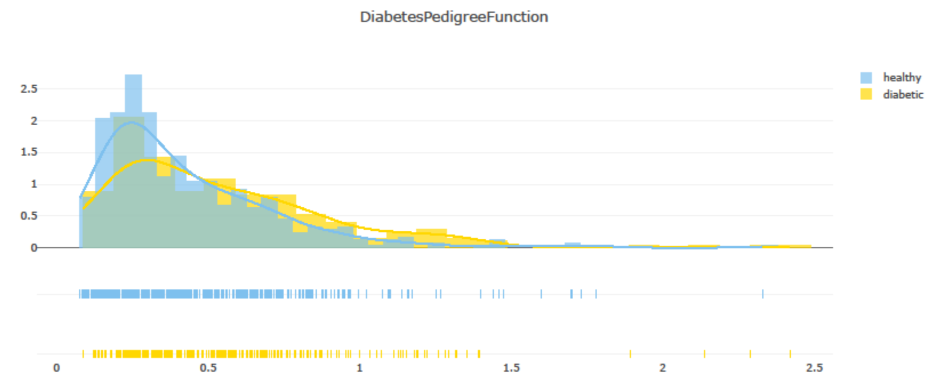
```
In [ ]: data.loc[(data['Outcome'] == 0 ) & (data['BMI'].isnull()), 'BMI'] = 30.1
        data.loc[(data['Outcome'] == 1 ) & (data['BMI'].isnull()), 'BMI'] = 34.3
```

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

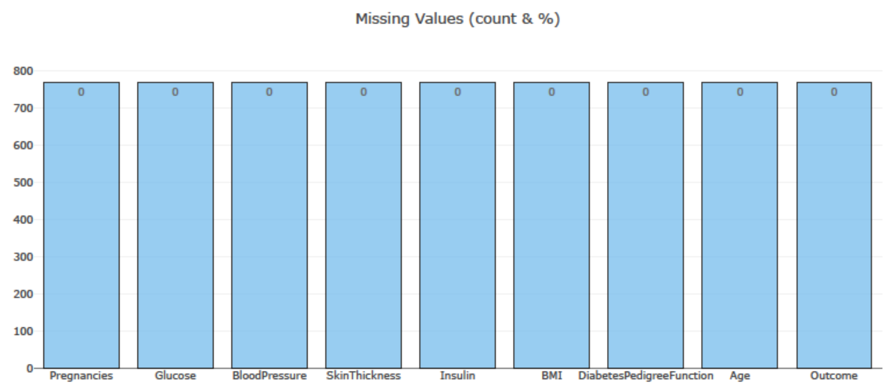
```
In [ ]: #plot distribution
        plot_distribution('Age', 0)
        plot_distribution('Pregnancies', 0)
        plot_distribution('DiabetesPedigreeFunction', 0)
```







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



All features are complete ! Now, we can create new features

4. New features (16) and EDA

Here, we define 3 plots functions

```
In [ ]: 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)
```

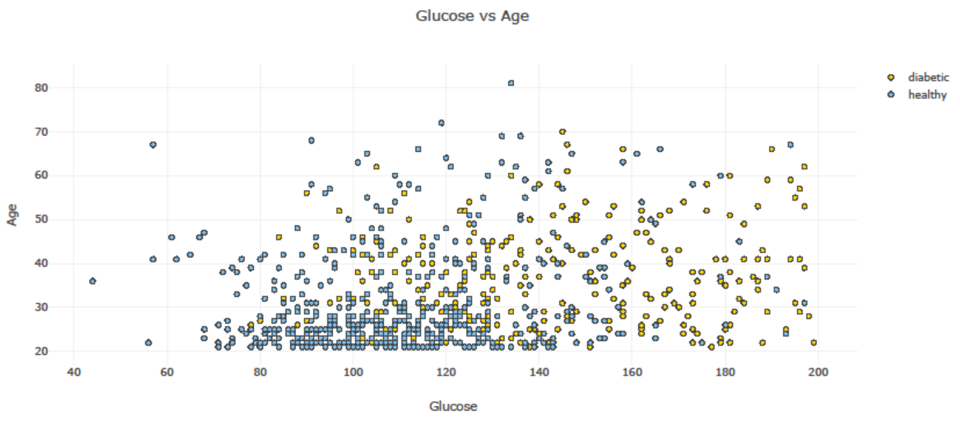
```
In [ ]: def barplot(var_select, sub) :
            tmp1 = data[(data['Outcome'] != 0)]
            tmp2 = data[(data['Outcome'] == 0)]
            tmp3 = pd.DataFrame(pd.crosstab(data[var_select],data['Outcome']), )
            tmp3['% diabetic'] = tmp3[1] / (tmp3[1] + tmp3[0]) * 100

            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 [ ]: # 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)
```

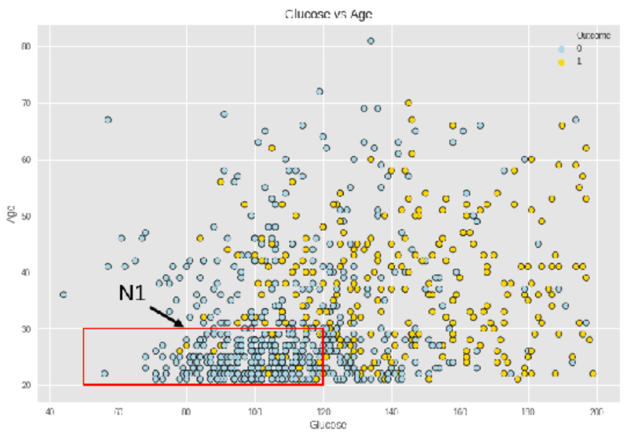
- **Glucose and Age**

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



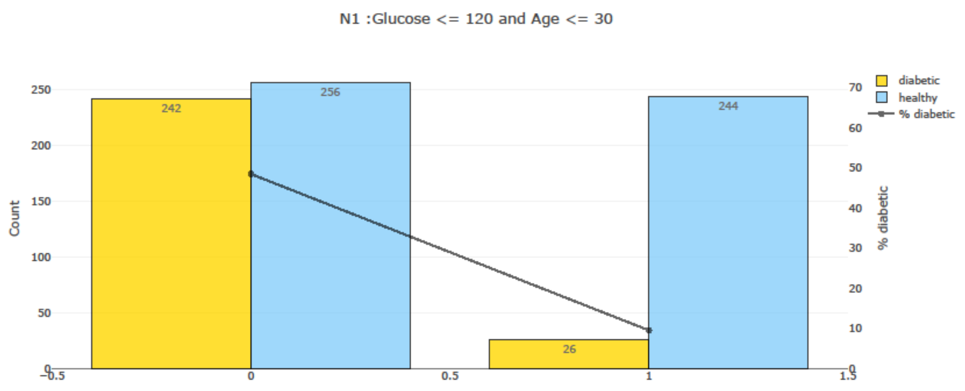
Healthy persons are concentrate with an age <= 30 and glucose <= 120

In [ ]:
```
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()
```

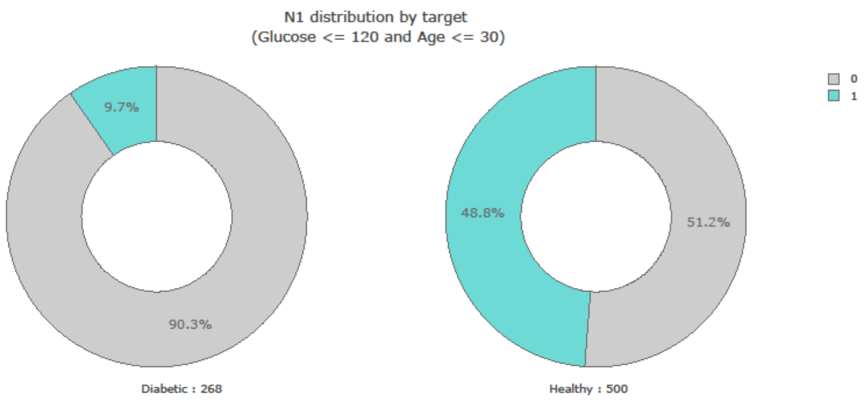


In [ ]:
```
data.loc[:,'N1']=0
data.loc[(data['Age']<=30) & (data['Glucose']<=120),'N1']=1
```

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



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



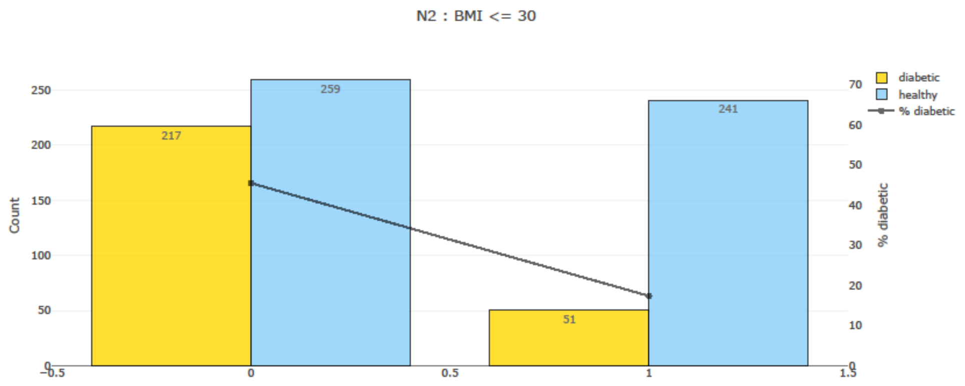
- **BMI **

According to wikipedia "The body mass index (BMI) or Quetelet index is a value derived from the mass (weight) and height of an individual. The BMI is defined as the body mass divided by the square of the body height, and is universally expressed in units of kg/m2, resulting from mass in kilograms and height in metres."

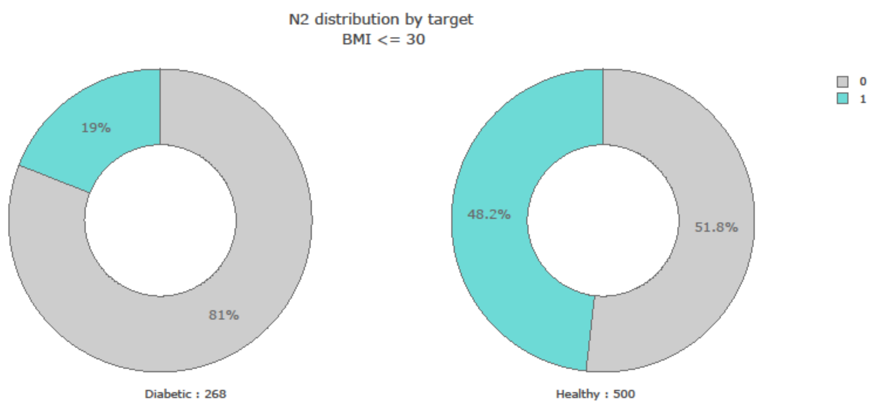
30 kg/m² is the limit to obesity

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

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

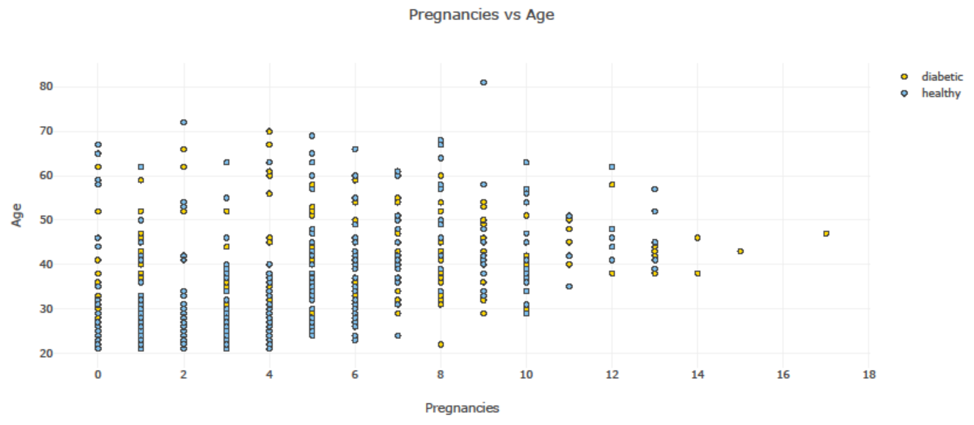


```
In [ ]: plot_pie('N2', 'BMI <= 30')
```

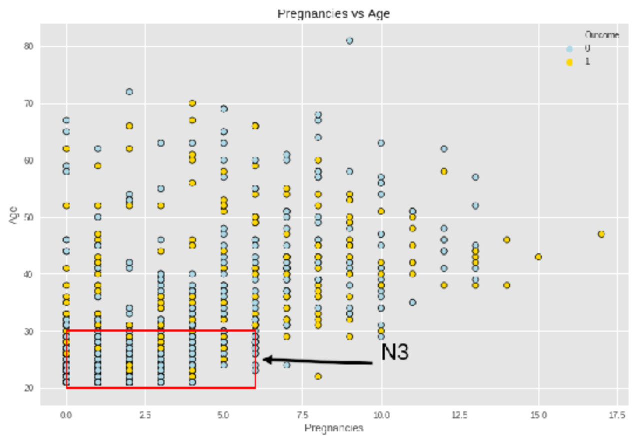


- **Pregnancies and Age**

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

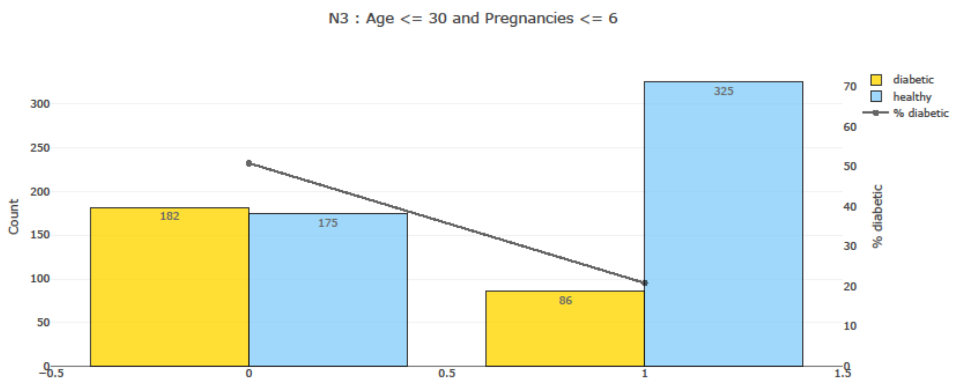


```
In [ ]: palette ={0 : 'lightblue', 1 : 'gold'}
        edgecolor = 'black'

        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()
```

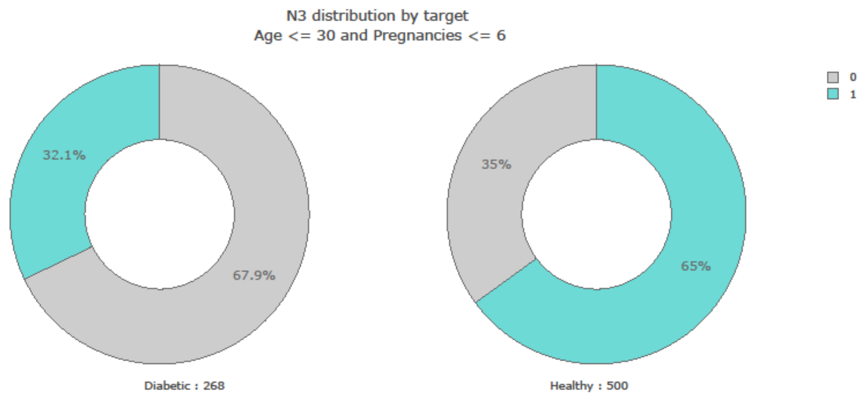


```
In [ ]: data.loc[:,'N3']=0
        data.loc[(data['Age']<=30) & (data['Pregnancies']<=6),'N3']=1
```

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

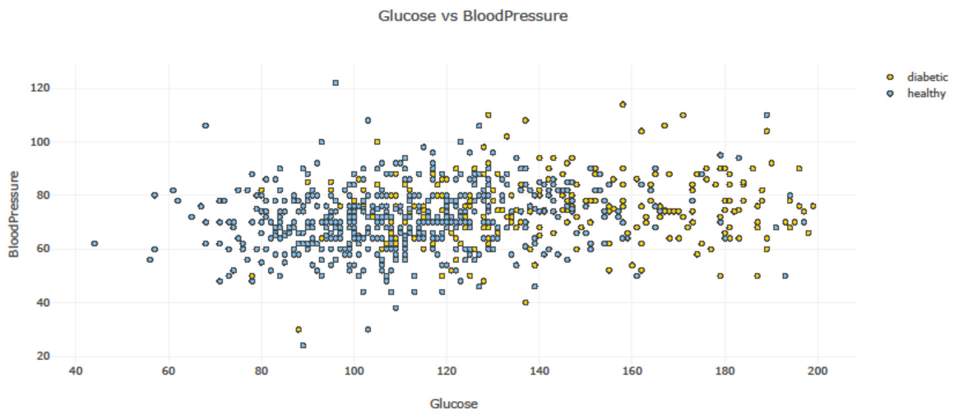


```
In [ ]: plot_pie('N3', 'Age <= 30 and Pregnancies <= 6')
```



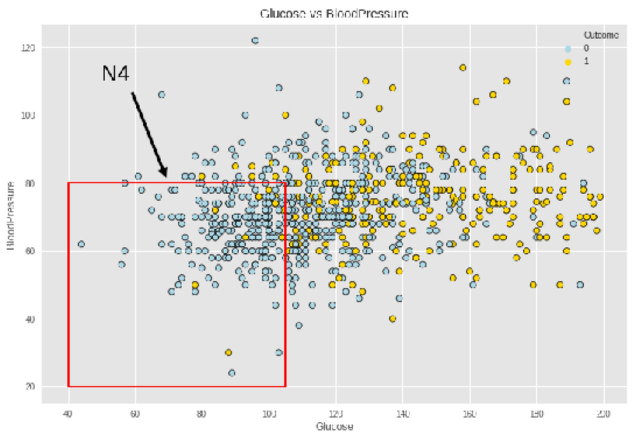
- **Glucose and BloodPressure**

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



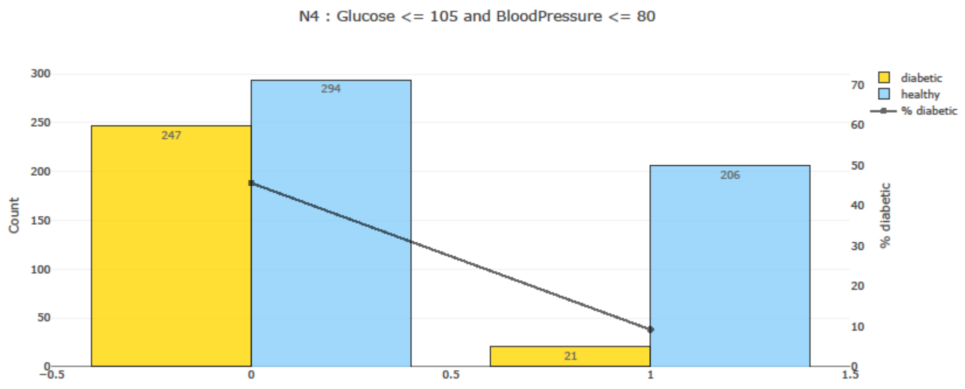
Healthy persons are concentrate with an blood pressure <= 80 and glucose <= 105

```
In [ ]: 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()
```

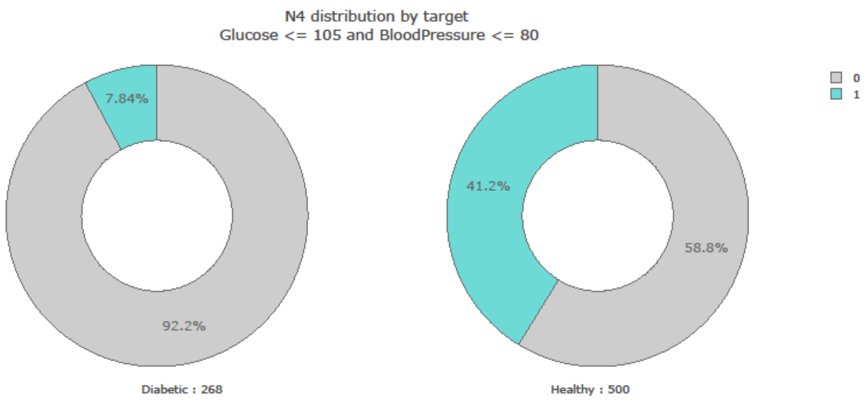


```
In [ ]: data.loc[:,'N4']=0
        data.loc[(data['Glucose']<=105) & (data['BloodPressure']<=80),'N4']=1
```

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



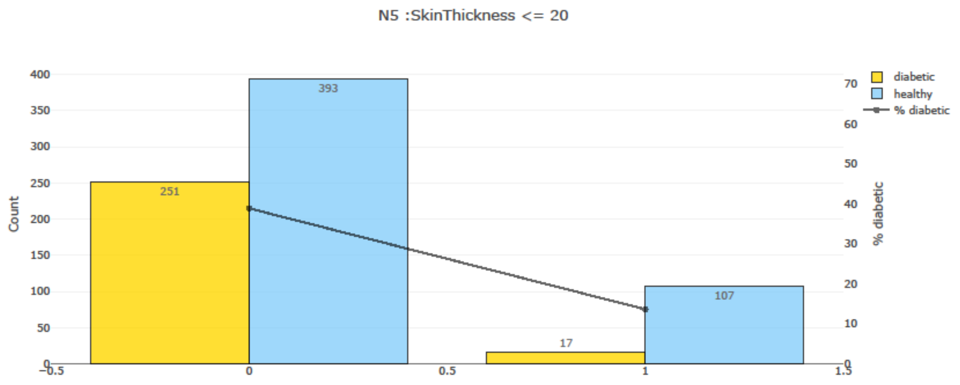
```
In [ ]: plot_pie('N4', 'Glucose <= 105 and BloodPressure <= 80')
```



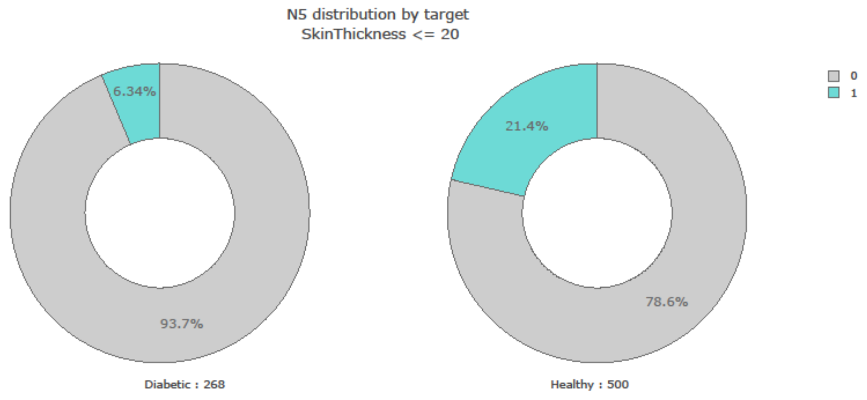
- **SkinThickness**

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

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

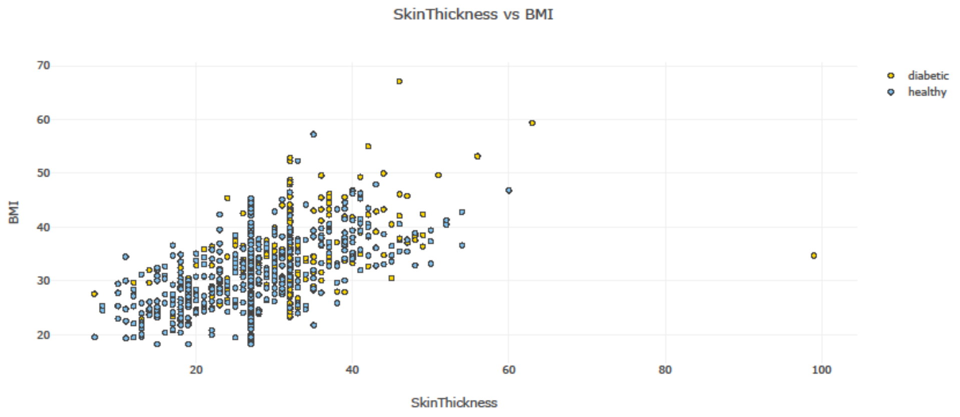


```
In [ ]: plot_pie('N5', 'SkinThickness <= 20')
```



- **SkinThickness and BMI**

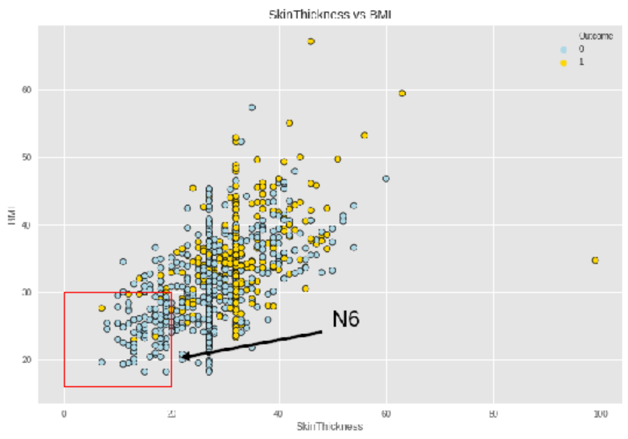
```
In [ ]: plot_feat1_feat2('SkinThickness','BMI')
```



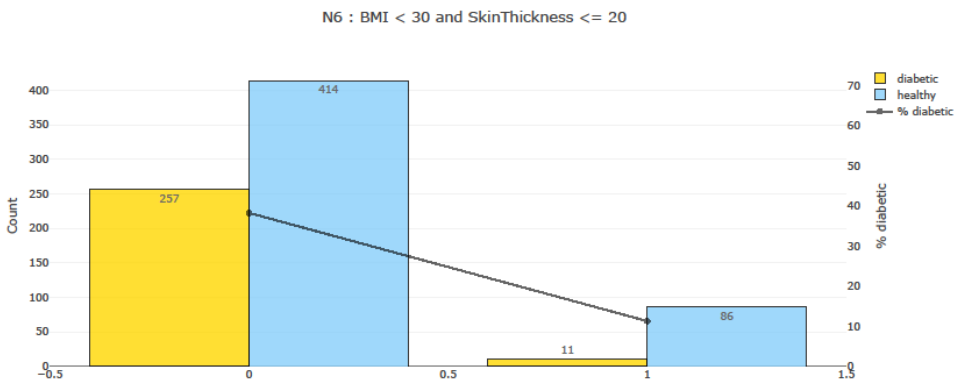
Healthy persons are concentrate with a BMI < 30 and skin thickness <= 20

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

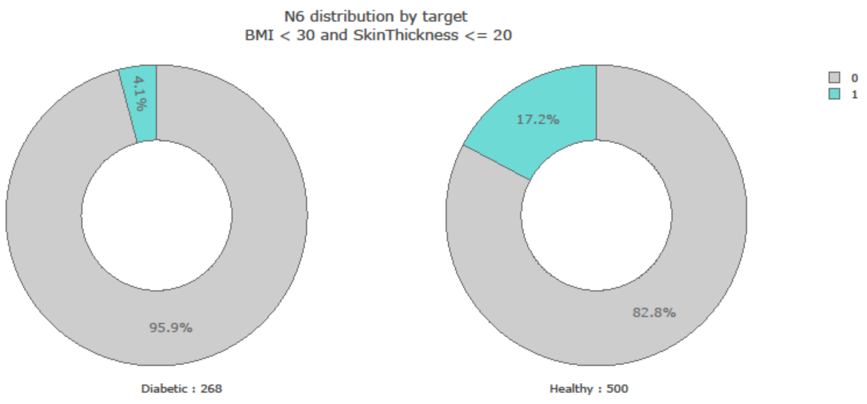
```
In [ ]: palette ={0 : 'lightblue', 1 : 'gold'}
        edgecolor = 'black'

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

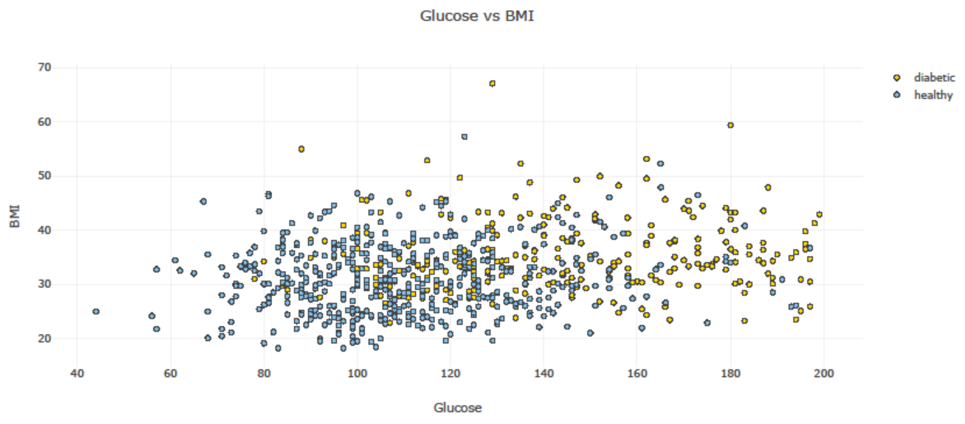


```
In [ ]: plot_pie('N6', 'BMI < 30 and SkinThickness <= 20')
```

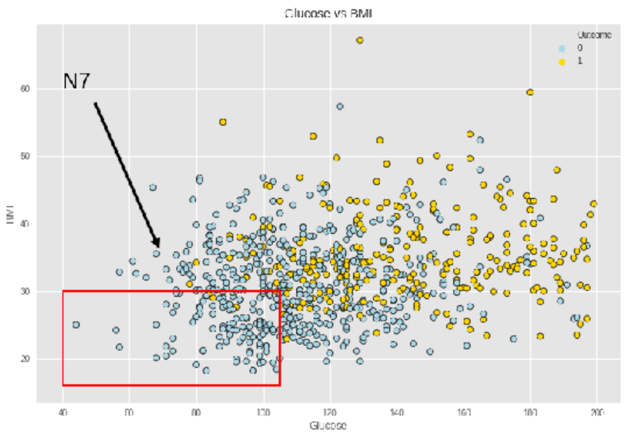


- **Glucose and BMI**

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

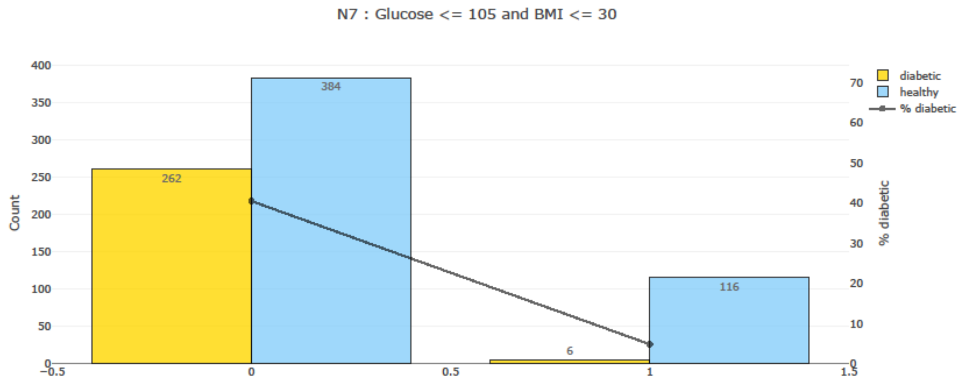


```
In [ ]: palette ={0 : 'lightblue', 1 : 'gold'}
        edgecolor = 'black'

        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()
```

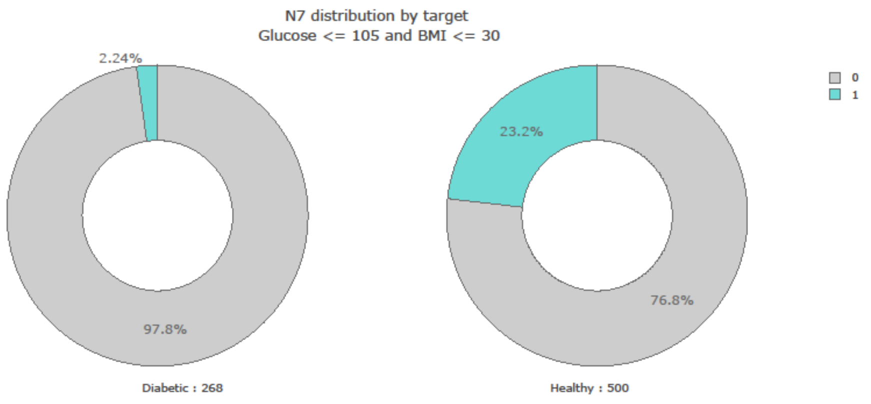


```
In [ ]: data.loc[:,'N7']=0
        data.loc[(data['Glucose']<=105) & (data['BMI']<=30),'N7']=1
```

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

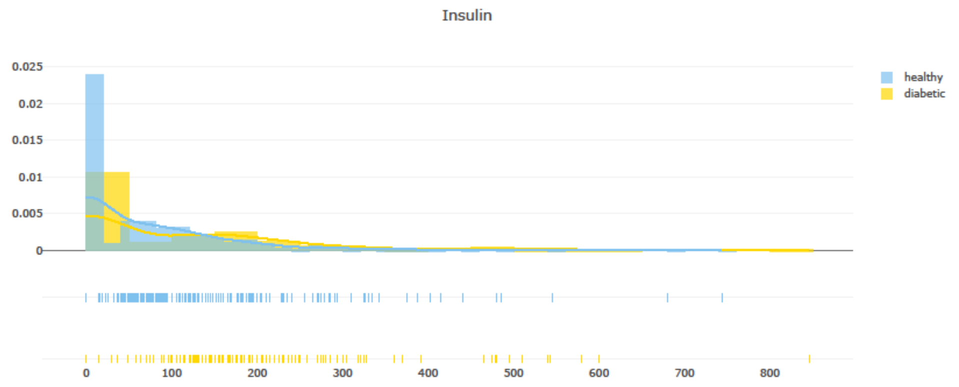


```
In [ ]: plot_pie('N7', 'Glucose <= 105 and BMI <= 30')
```



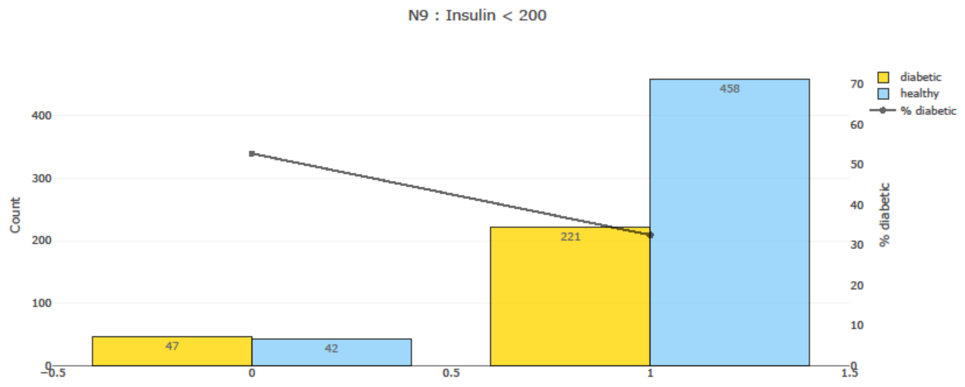
- **Insulin**

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

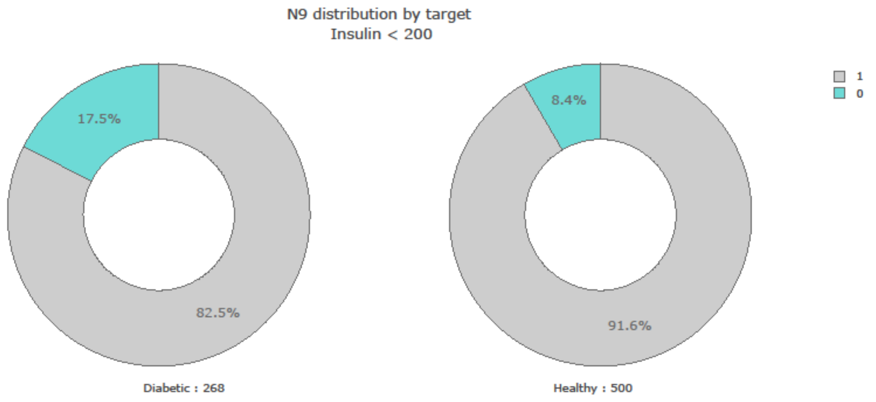


```
In [ ]: data.loc[:,'N9']=0
        data.loc[(data['Insulin']<200),'N9']=1
```

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



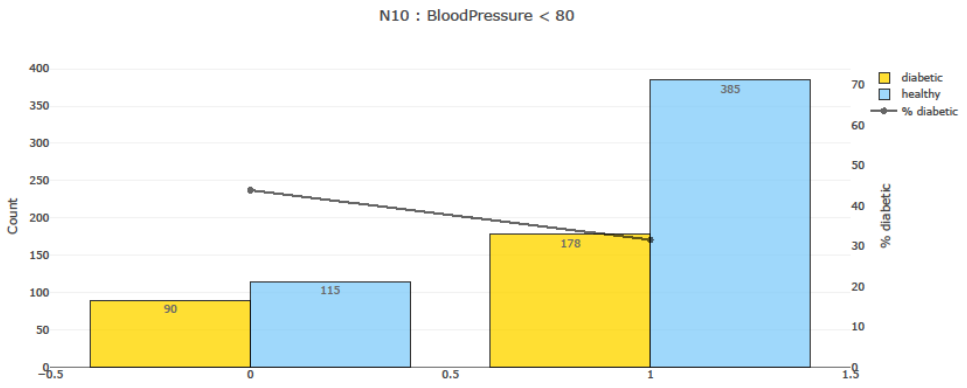
```
In [ ]: plot_pie('N9', 'Insulin < 200')
```



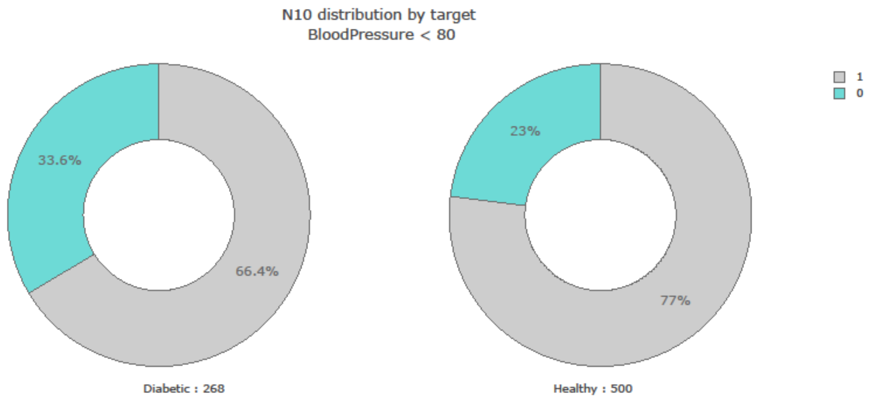
- **BloodPressure**

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

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

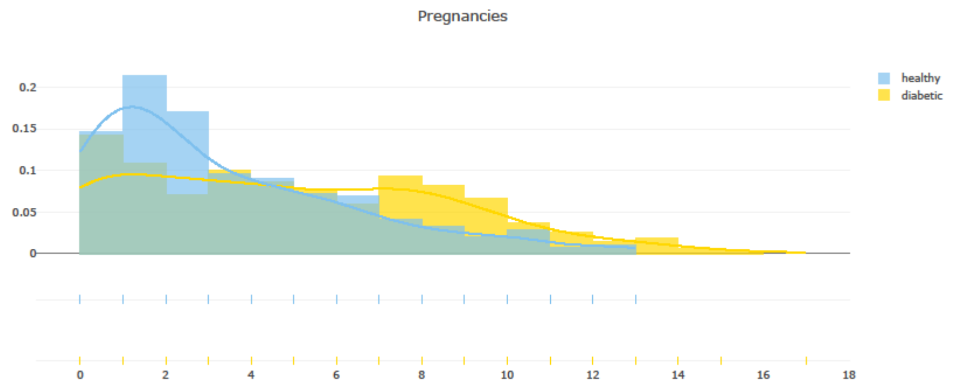


```
In [ ]: plot_pie('N10', 'BloodPressure < 80')
```



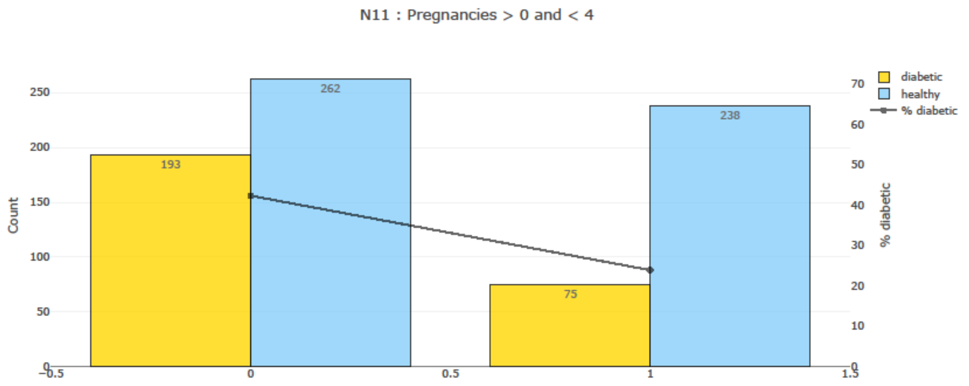
- **Pregnancies**

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

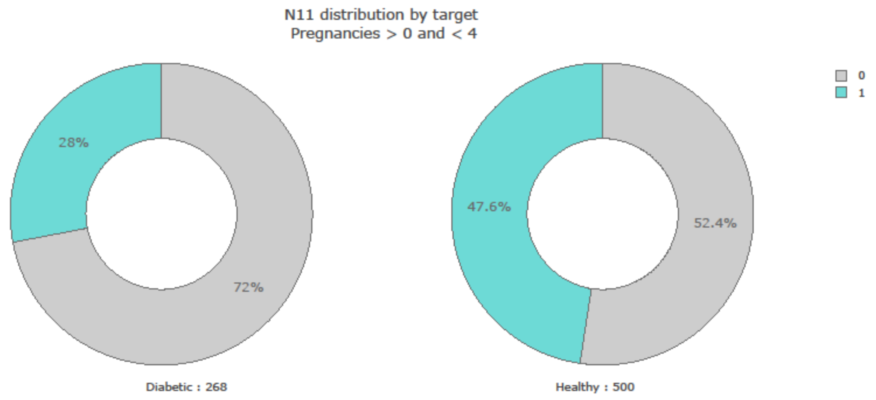


```
In [ ]: data.loc[:,'N11']=0
        data.loc[(data['Pregnancies']<4) & (data['Pregnancies']!=0) ,'N11']=1
```

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



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



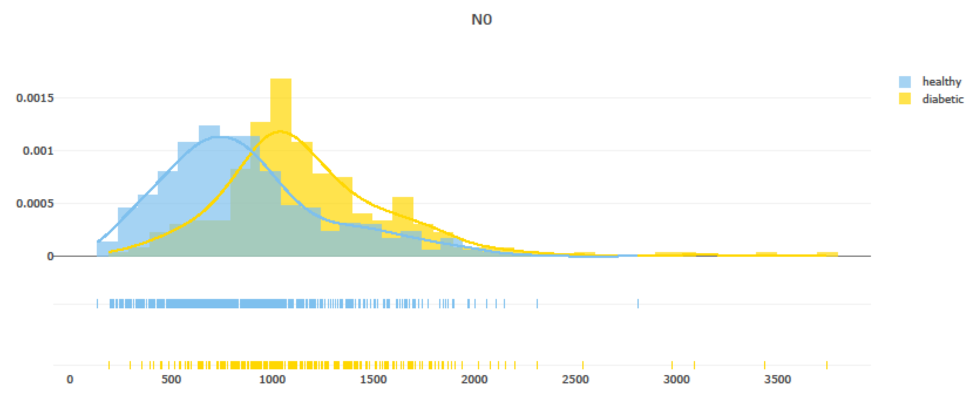
- **Others**

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

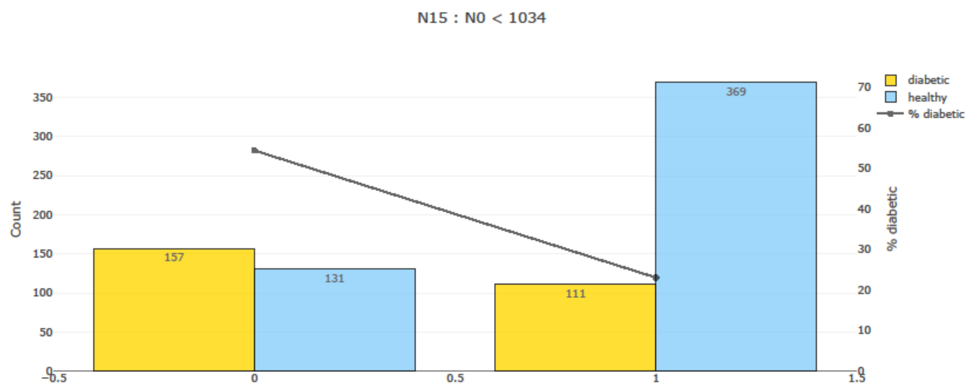
- ** Did you watch Inception ? ** Here is the same! It's not a dream in a dream but a new feature extract from a new feature

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

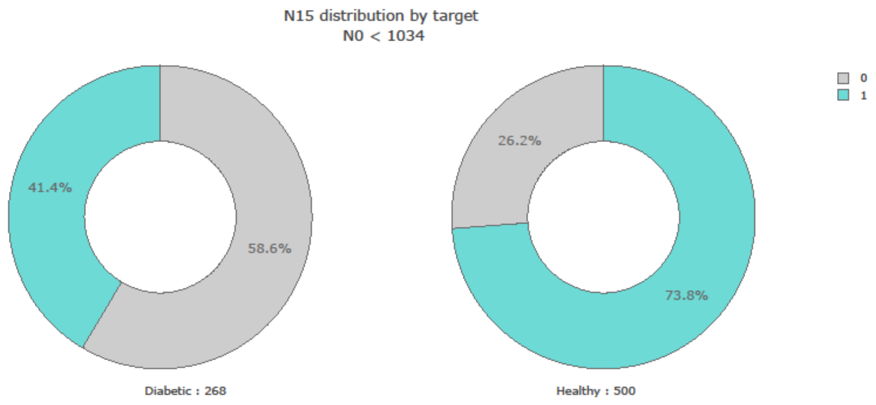


```
In [ ]: data.loc[:,'N15']=0
        data.loc[(data['N0']<1034) ,'N15']=1
```

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



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



5. Prepare dataset

5.1. StandardScaler and LabelEncoder

- ** StandardScaler** :

Standardize features by removing the mean and scaling to unit variance :

Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the set. Mean and standard deviation are then stored to be used on later data using the transform method.

Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual features do not more or less look like standard normally distributed data (e.g. Gaussian with 0 mean and unit variance).

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

Bellow we encode the data to feed properly to our algorithm

In [ ]:
```
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")
```

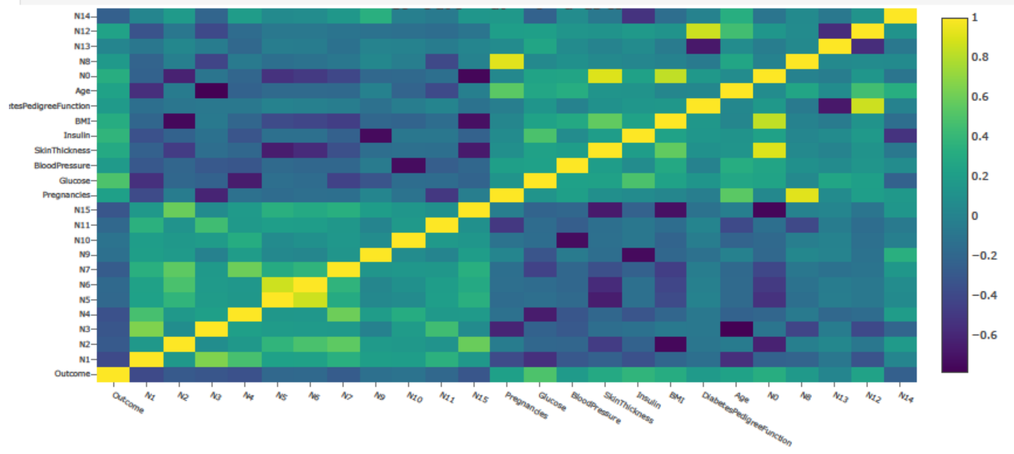
Now, we can compute correlation matrix

5.2. Correlation Matrix

A **correlation matrix** is a table showing correlation coefficients between sets of variables. Each random variable (Xi) in the table is correlated with each of the other values in the table (Xj). This allows you to see which pairs have the highest correlation.

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



5.3. X and y

We define X and y :

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

5.4. Model Performance

To measure the performance of a model, we need several elements :

This part is essential

- **Confusion matrix** : also known as the error matrix, allows visualization of the performance of an algorithm :
  - true positive (TP) : Diabetic correctly identified as diabetic
  - true negative (TN) : Healthy correctly identified as healthy
  - false positive (FP) : Healthy incorrectly identified as diabetic

- false negative (FN) : Diabetic incorrectly identified as healthy

- **Metrics ** :
  - Accuracy : (TP +TN) / (TP + TN + FP +FN)
  - Precision : TP / (TP + FP)
  - Recall : TP / (TP + FN)
  - F1 score : 2 x ((Precision x Recall) / (Precision + Recall))
- **Roc Curve** : The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.

- **Precision Recall Curve** : shows the tradeoff between precision and recall for different threshold

To train and test our algorithm we'll use cross validation K-Fold

In k-fold cross-validation, the original sample is randomly partitioned into k equal sized subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining k − 1 subsamples are used as training data. The cross-validation process is then repeated k times, with each of the k subsamples used exactly once as the validation data. The k results can then be averaged to produce a single estimation. The advantage of this method over repeated random sub-sampling is that all observations are used for both training and validation, and each observation is used for validation exactly once.

Bellow we define a stylized report with Plotly

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

We can complete model performance report with a table contain all results by fold

In [ ]:
```
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 - Accuracy = 89.8%

** LightGBM** is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with the following advantages:

- Faster training speed and higher efficiency.
- Lower memory usage.
- Better accuracy.
- Support of parallel and GPU learning.
- Capable of handling large-scale data.

To find the best hyperparameters, we'll use Random Search CV.

Random search is a technique where random combinations of the hyperparameters are used to find the best solution for the built model. Generally RS is more faster and accurate than GridSearchCV who calculate all possible combinations. With Random Grid we specify the number of combinations that we want

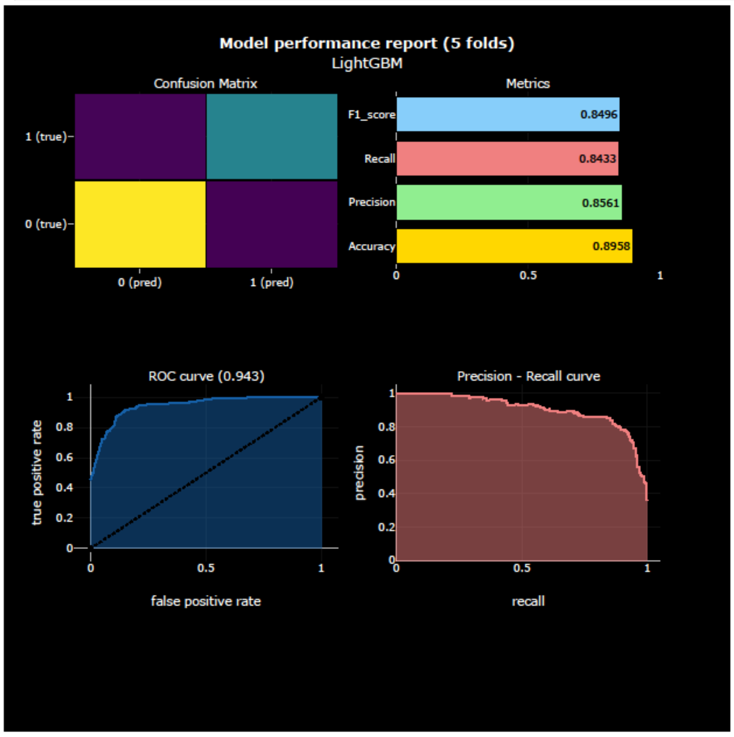
- **LightGBM : Hyperparameters ** :
  - learning_rate : This determines the impact of each tree on the final outcome. GBM works by starting with an initial estimate which is updated using the output of each tree. The learning parameter controls the magnitude of this change in the estimates
  - n_estimators : number of trees (or rounds)
  - num_leaves : number of leaves in full tree, default: 31
  - min_child_samples : minimal number of data in one leaf. Can be used to deal with over-fitting
  - min_child_weight : minimal sum hessian in one leaf.
  - subsample : randomly select part of data without resampling
  - max_depth : It describes the maximum depth of tree. This parameter is used to handle model overfitting.
  - colsample_bytree : LightGBM will randomly select part of features on each iteration if colsample_bytree smaller than 1.0. For example, if you set it to 0.8, LightGBM will select 80% of features before training each tree
  - reg_alpha : regularization
  - reg_lambda : regularization
  - early_stopping_rounds : This parameter can help you speed up your analysis. Model will stop training if one metric of one validation data doesn't improve in last early_stopping_round rounds. This will reduce excessive iterations

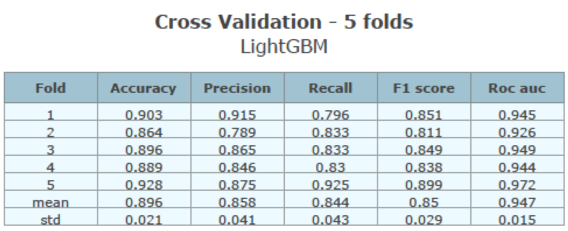
In [ ]:
```
random_state=42

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
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 1500 out of 1500 | elapsed:  1.9min finished
```

```
In [ ]: model_performance(lgbm_clf, 'LightGBM')
        scores_table(lgbm_clf, 'LightGBM')
```





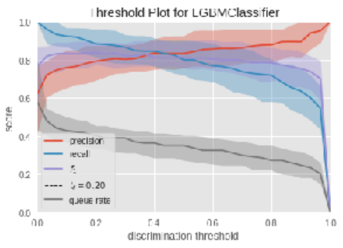
Cross Validation - 5 folds
LightGBM

Fold	Accuracy	Precision	Recall	F1 score	Roc auc
1	0.903	0.915	0.796	0.851	0.945
2	0.864	0.789	0.833	0.811	0.926
3	0.896	0.865	0.833	0.849	0.949
4	0.889	0.846	0.83	0.838	0.944
5	0.928	0.875	0.925	0.899	0.972
mean	0.896	0.858	0.844	0.85	0.947
std	0.021	0.041	0.043	0.029	0.015

6.2. LightGBM - Discrimination Threshold

- **Discrimination Threshold :** A visualization of precision, recall, f1 score, and queue rate with respect to the discrimination threshold of a binary classifier. The discrimination threshold is the probability or score at which the positive class is chosen over the negative class

```
In [ ]: visualizer = DiscriminationThreshold(lgbm_clf)

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



6.3. GridSearch + LightGBM & KNN- Accuracy = 90.6%

We obtain a really good result but we can beat 90% with adding a KNeighborsClassifier to LightGBM (Voting Classifier)

- **KNeighborsClassifier :** KNeighborsClassifier implements learning based on the k nearest neighbors of each query point, where k is an integer value specified by the user.
- **VotingClassifier :** VotingClassifier is a meta-classifier for combining similar or conceptually different machine learning classifiers for classification via majority or plurality voting

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

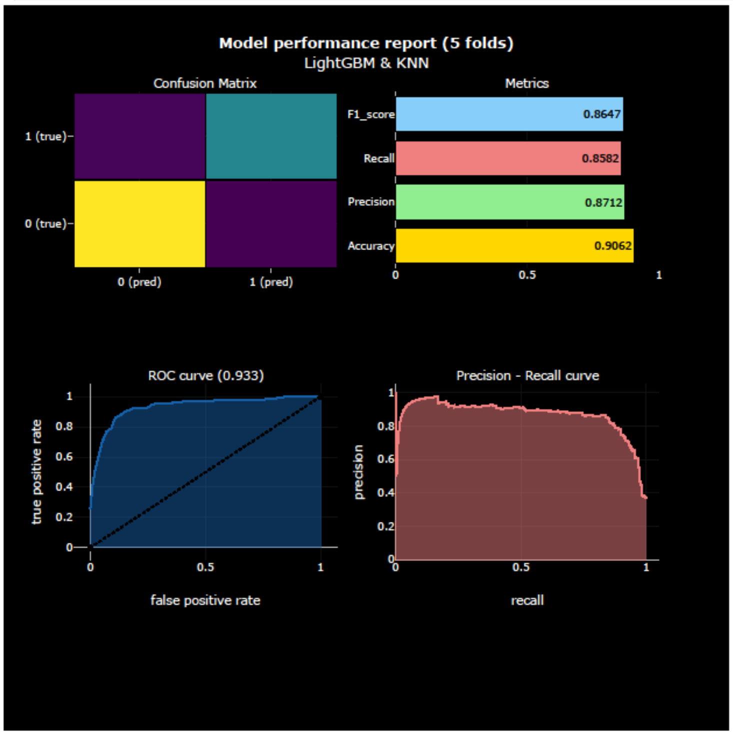
```
In [ ]: knn_clf = KNeighborsClassifier()

        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_))
```

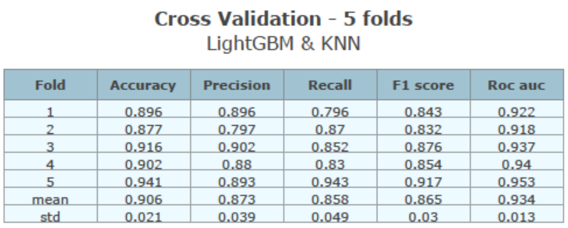
```
Best Score:0.90625
Best Parameters: {'knn__n_neighbors': 25}
```

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

```
In [ ]: knn_clf = KNeighborsClassifier(n_neighbors = 25)

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

```
In [ ]: model_performance(voting_clf, 'LightGBM & KNN')
        scores_table(voting_clf, 'LightGBM & KNN')
```



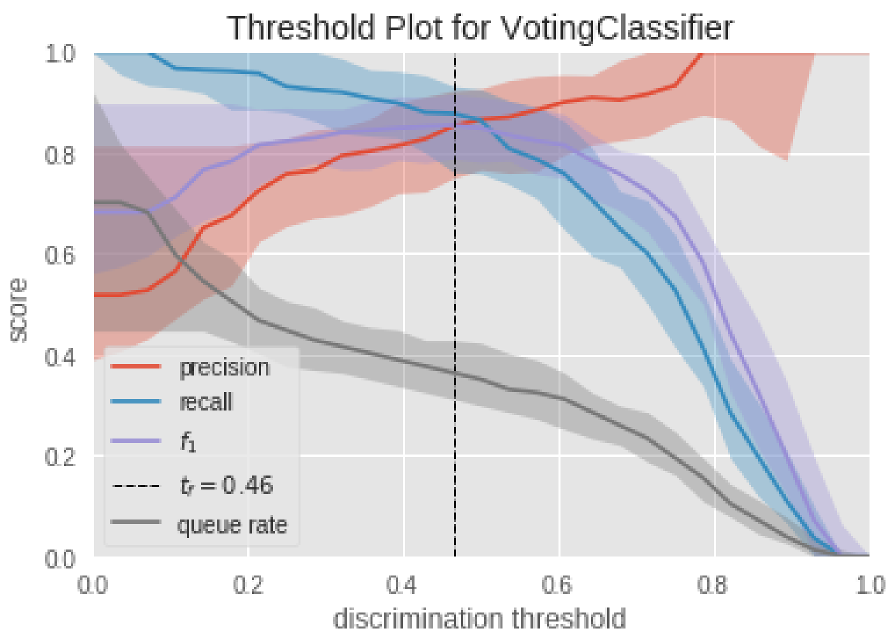


Cross Validation - 5 folds
LightGBM & KNN

Fold	Accuracy	Precision	Recall	F1 score	Roc auc
1	0.896	0.896	0.796	0.843	0.922
2	0.877	0.797	0.87	0.832	0.918
3	0.916	0.902	0.852	0.876	0.937
4	0.902	0.88	0.83	0.854	0.94
5	0.941	0.893	0.943	0.917	0.953
mean	0.906	0.873	0.858	0.865	0.934
std	0.021	0.039	0.049	0.03	0.013

6.4. LightGBM & KNN - Discrimination Threshold

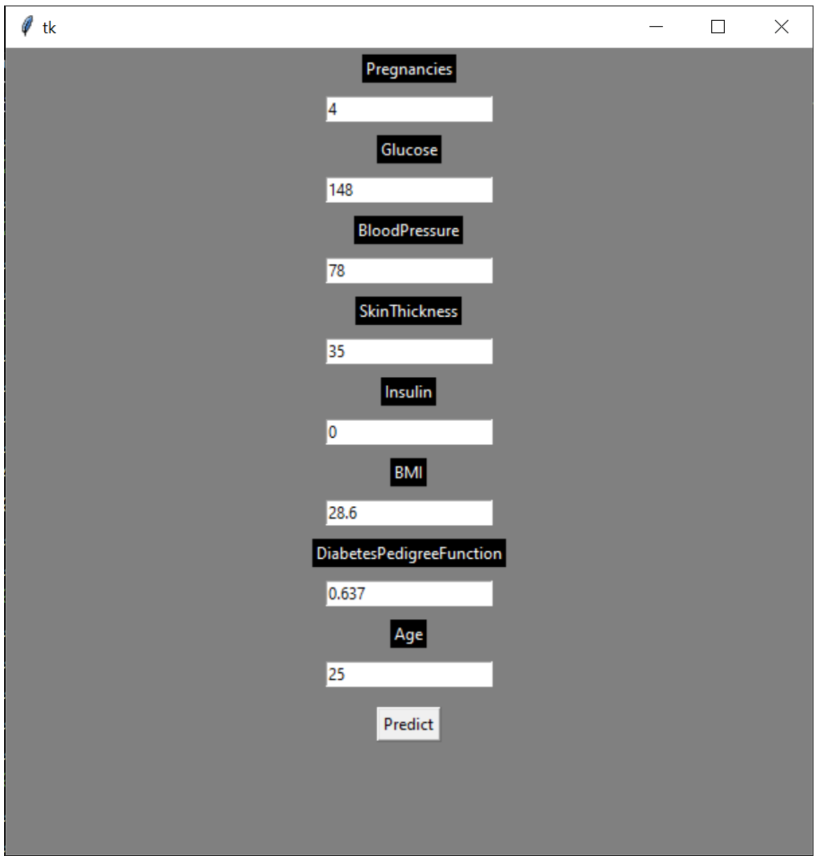
```
In [ ]: visualizer = DiscriminationThreshold(voting_clf)

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



7. Results:

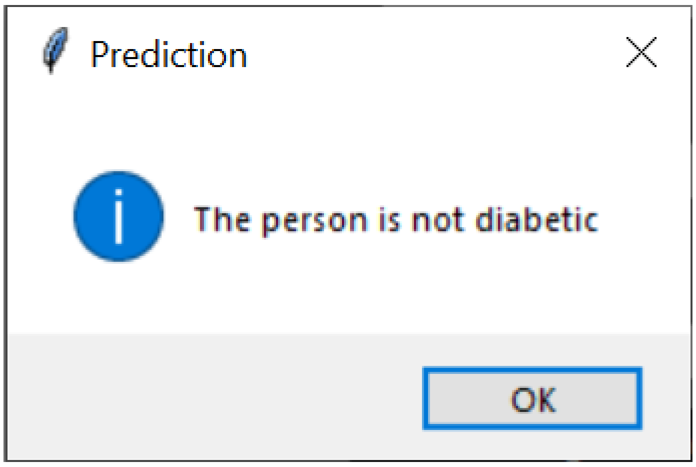
7.1. Setup:

- run `‘diabetes_predictorGUI.py’` as the base file to start.
- Fill the data in respective location.
- Hit `‘predict’`


tk

Pregnancies
4
Glucose
148
BloodPressure
78
SkinThickness
35
Insulin
0
BMI
28.6
DiabetesPedigreeFunction
0.637
Age
25

7.2. Output:



Prediction

The person is not diabetic

7.3. Observation:

It is really important what dataset we provide to it, as the dataset is the factor that the model gets trained.

Our model is biased towards "Glucose ", "Insulin" and "DiabetesPedigreeFunction".

8. References:

- https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc
- https://en.wikipedia.org/wiki/Body_mass_index
- http://rasbt.github.io/mlxtend/user_guide/classifier/EnsembleVoteClassifier/
- https://www.news-medical.net/health/What-is-Diabetes.aspx
- https://lightgbm.readthedocs.io/en/latest/Parameters-Tuning.html
- http://ogrisel.github.io/scikit-learn.org/sklearn-tutorial/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
- https://www.scikit-yb.org/en/latest/api/classifier/threshold.html
- https://www.analyticsindiamag.com/why-is-random-search-better-than-grid-search-for-machine-learning/
- https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html
- https://scikit-learn.org/stable/modules/preprocessing_targets.html#preprocessing-targets
- https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
- https://twitter.com/bearda24
- https://www.slideshare.net/DhianaDevaRocha/qcon-rio-machine-learning-for-everyone
- https://medium.com/@sebastiannorena/some-model-tuning-methods-bfef3e6544f0
- https://www.niddk.nih.gov/health-information/diabetes/overview/what-is-diabetes
- https://en.wikipedia.org/wiki/Pima_people