VIETNAM NATIONAL UNIVERSITY-HO CHI MINH CITY

UNIVERSITY OF SCIENCE

FACULTY OF INFORMATION TECHNOLOGY

FINAL PROJECT REPORT

Student	**:**	**20C14001 – Le Duong Tuan Anh**
Teacher	**:**	**Dr. Le Thi Nhan**
Class	**:**	**Master of Information System**
Subject	**:**	**Data Mining**

Ho Chi Minh city, July 2021

Contents

1 General Information

Student Information

Student ID	Full Name	Email
20C14001	Le Duong Tuan Anh	leduongtuananh97@gmail.com

Report Information

This report is a summary of the **Final Project** based on **Diabetes prediction with given Pima Indians medical details.**

2 Problem Statement

In this project, I use dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases, which contains data between relationship between some medical details and the result on Diabetes.

The objective of the dataset is to diagnostically predict whether a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

The dataset consists of **several medical predictor variables** and **one target variable**, ***Outcome***.

Predictor variables includes *the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.*

3 Data Description

Total instances (records): 768

Total columns: 9 (8 medical attributes + 1 Outcome Class (has diabetes or not)).

Attribute's detail:

Column	Type	Datatype	Has missing value?
Pregnancies	Interval-scaled	Int64	No
Glucose	Interval-scaled	Int64	No
BloodPressure	Interval-scaled	Int64	No
SkinThickness	Interval-scaled	Int64	No
Insulin	Interval-scaled	Int64	No
BMI	Interval-scaled	Float	No
DiabetesPedigreeFunction	Interval-scaled	Float	No
Age	Interval-scaled	Int64	No
Outcome	Categorical Data Binary (0/1)	Int64	No

Attribute's meaning:

Pregnancies: Number of times pregnant

Glucose: Plasma Glucose Concentration.

BloodPressure: Diastolic Blood Pressure.

SkinThickness: Estimate body fat.

Insulin: 2-Hour Serum Insulin.

BMI: Body Mass Index.

DiabetesPedigreeFunction: information about diabetes history in relatives and genetics.

Age: Age (years).

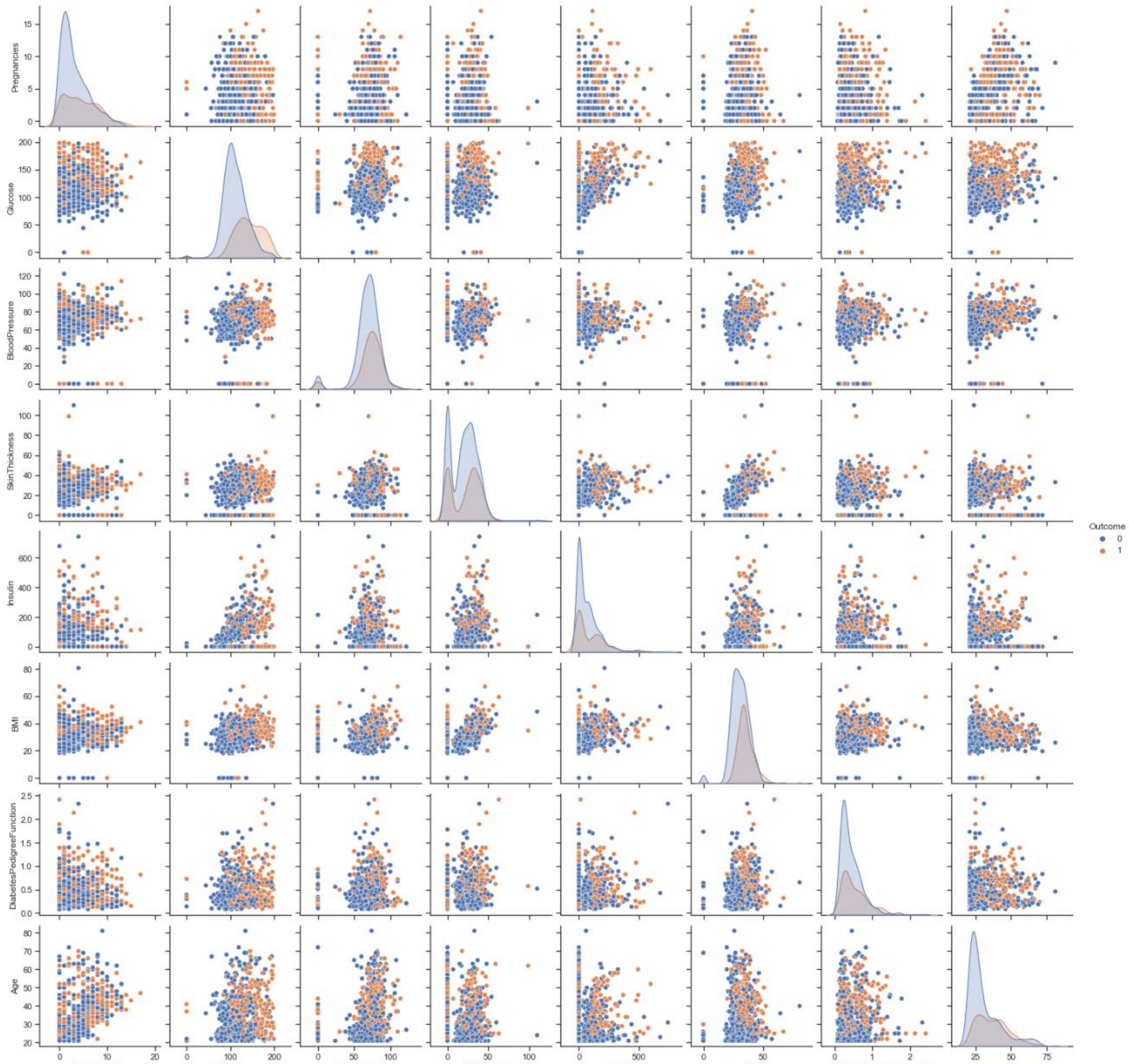
Outcome: 0 = Diabetic, 1 = Not Diabetic

4 Data Understanding

4.1 Description

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

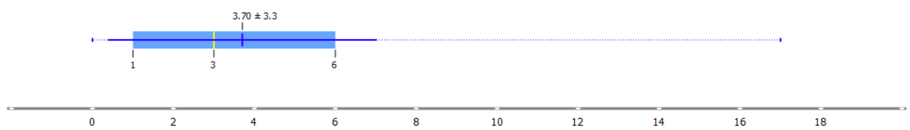
4.2 Overview by ScatterPlot

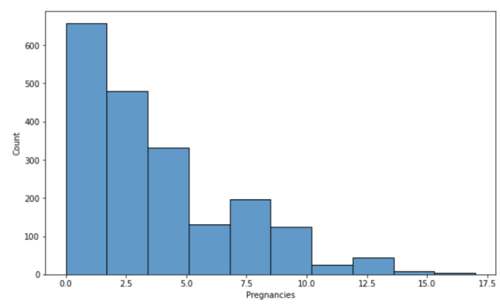


4.3 Statistical Information

2.1. Pregnancies

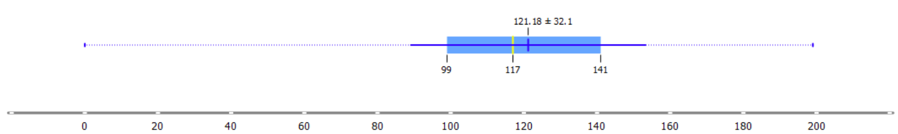
	count	mean	std	min	25%	50%	75%	max
Pregnancies	2000.0	3.7035	3.306063	0.0	1.0	3.0	6.0	17.0

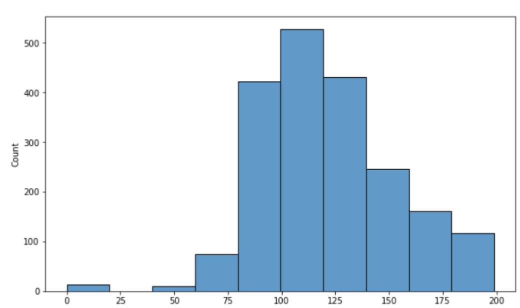




2.2. Glucose

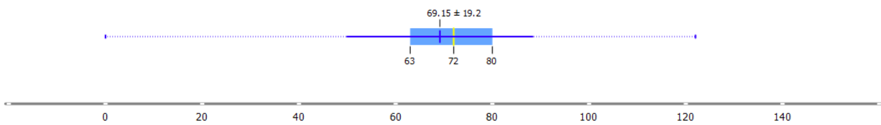
	count	mean	std	min	25%	50%	75%	max
Glucose	2000.0	121.1825	32.068636	0.0	99.0	117.0	141.0	199.0

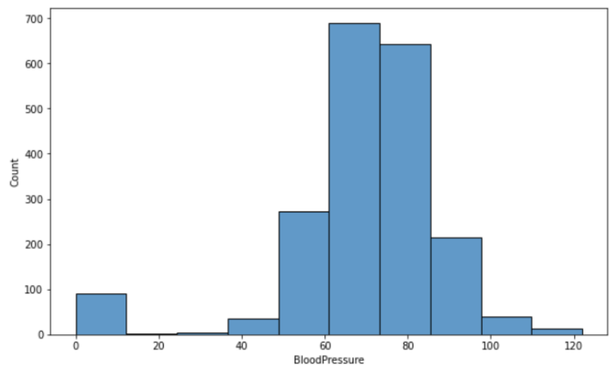




2.3. BloodPressure

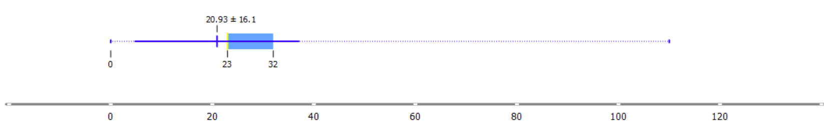
	count	mean	std	min	25%	50%	75%	max
BloodPressure	2000.0	69.1455	19.188315	0.0	63.5	72.0	80.0	122.0

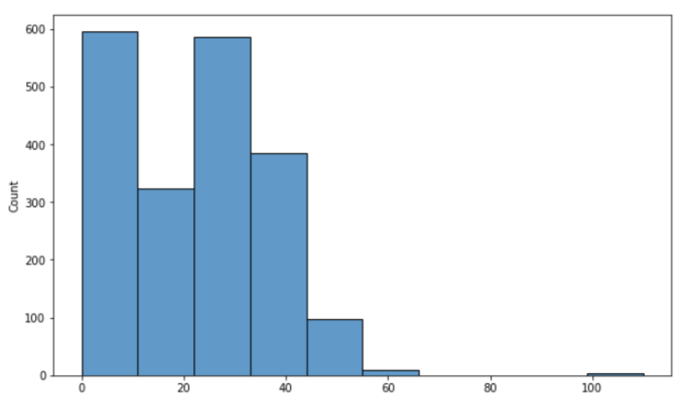




2.4. SkinThickness

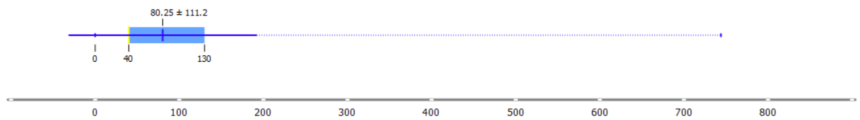
	count	mean	std	min	25%	50%	75%	max
SkinThickness	2000.0	20.935	16.103243	0.0	0.0	23.0	32.0	110.0

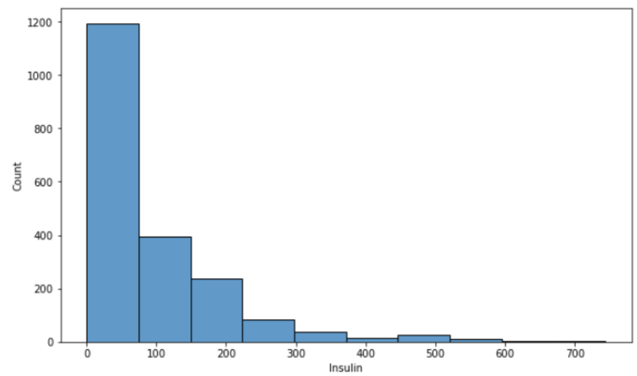




2.5. Insulin

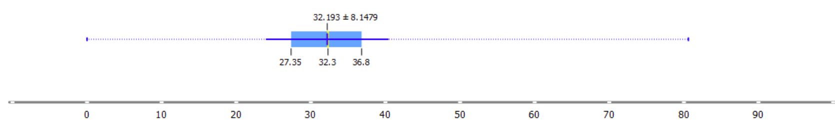
	count	mean	std	min	25%	50%	75%	max
Insulin	2000.0	80.254	111.180534	0.0	0.0	40.0	130.0	744.0

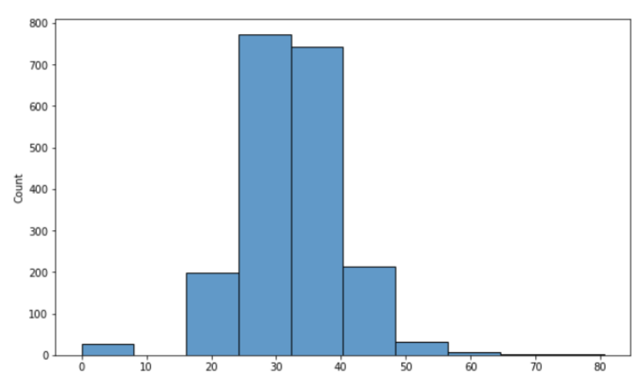




2.6. BMI

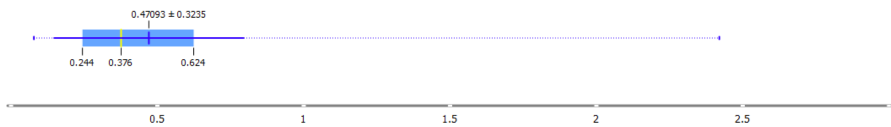
	count	mean	std	min	25%	50%	75%	max
BMI	2000.0	32.193	8.149901	0.0	27.375	32.3	36.8	80.6

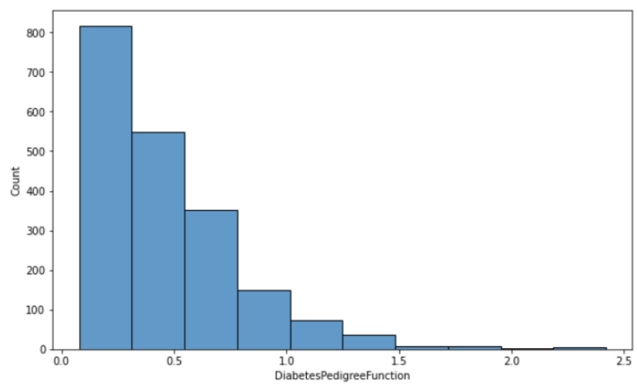




2.7. DiabetesPedigreeFunction

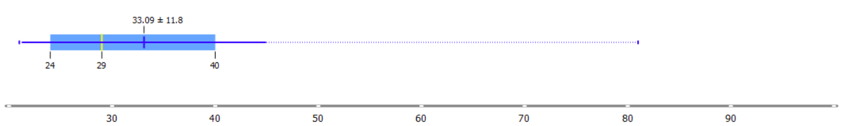
	count	mean	std	min	25%	50%	75%	max
DiabetesPedigreeFunction	2000.0	0.47093	0.323553	0.078	0.244	0.376	0.624	2.42

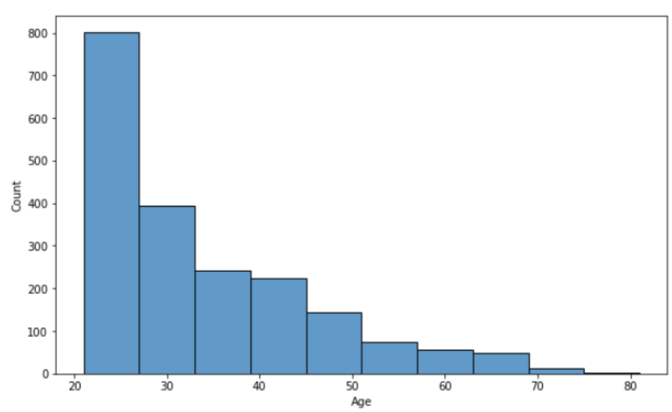




2.8. Age

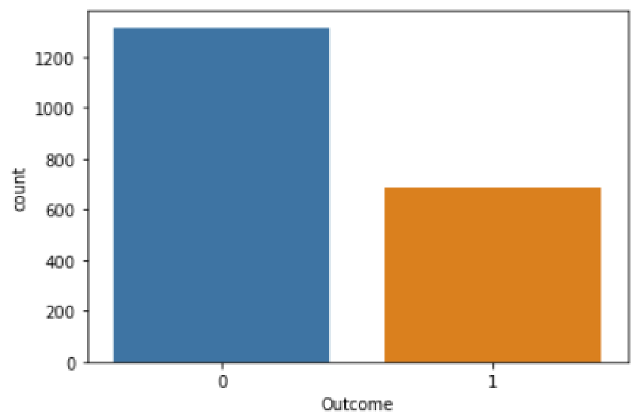
	count	mean	std	min	25%	50%	75%	max
Age	2000.0	33.0905	11.786423	21.0	24.0	29.0	40.0	81.0





2.9. Outcome

Outcome	
0	1316
1	684



5 Data Preprocessing

As we don't have any null-values, so we will check on zero-values and remove outliers (if needed).

5.1. Filling Zero-Values

column	total_zeros_value
Pregnancies	111
Glucose	5
BloodPressure	35
SkinThickness	227
Insulin	374
BMI	11
DiabetesPedigreeFunction	0
Age	0
Outcome	500

The **Pregnancies** column can be zero, as the gender might be male. There are 5 columns has zeros values: **Glucose, BloodPressure, SkinThickness, Insulin and BMI**. For example, SkinThickness cannot be zero, and BMI also. The other medical details cannot be zero, at least there must be very small values.

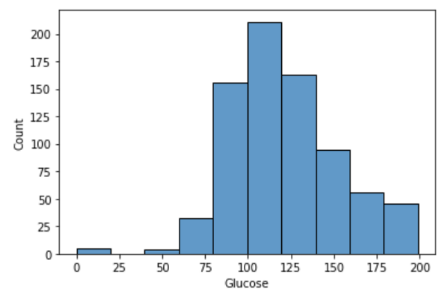
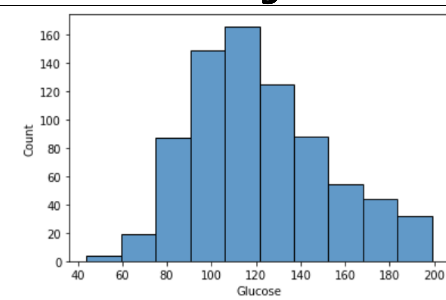
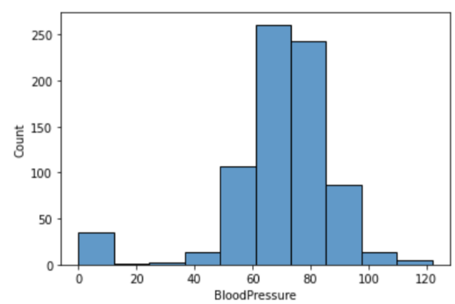
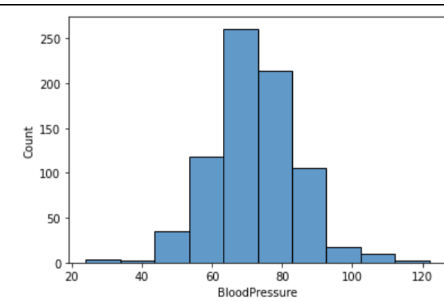
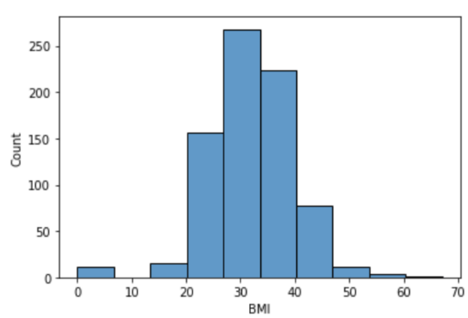
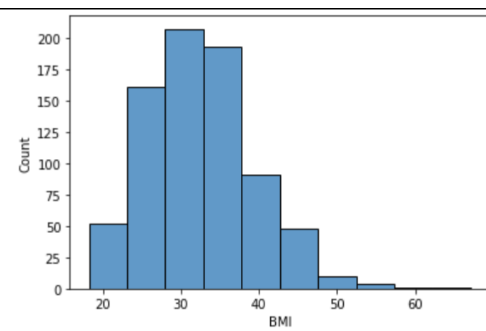
So, let's apply different type of zero-filling methods here.

5.1.1. Fill by Mean method

We just fill by mean on columns that have not much zero-values.

The applied columns: Glucose, BloodPressure, BMI.

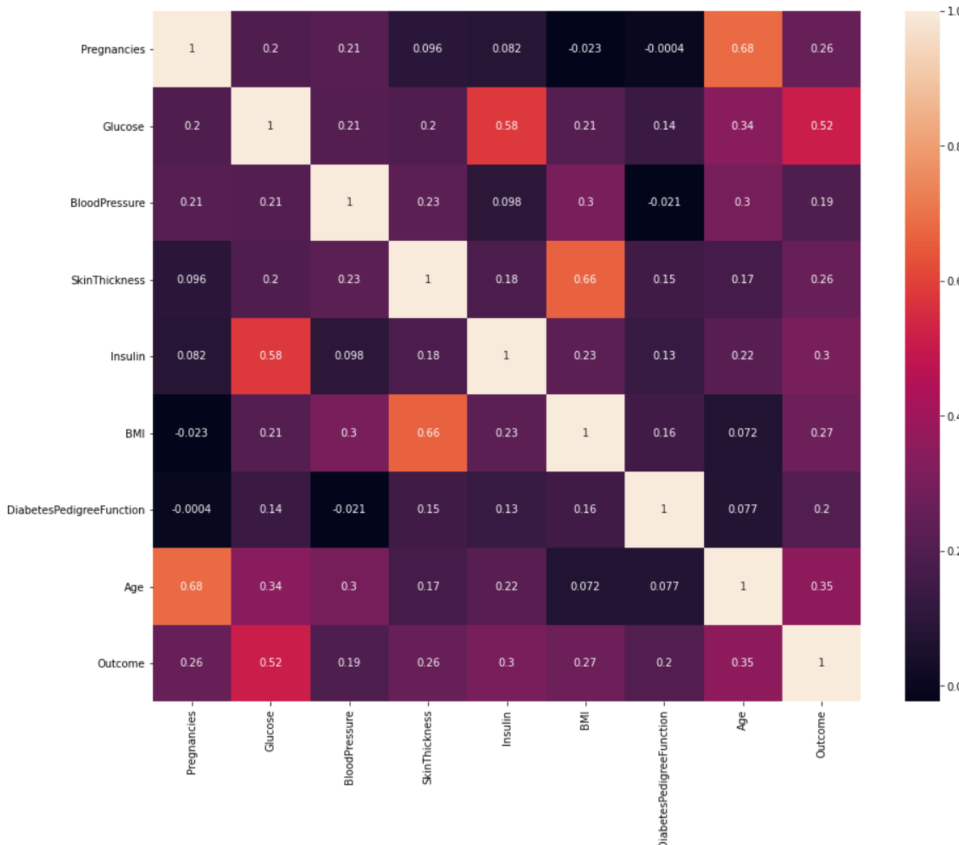
```python
# Glucose, BloodPressure, BMI can't be zero as well
# Let's fill by mean
for col in ['Glucose', 'BloodPressure', 'BMI']:
    data[col] = data[col].replace(value=data[col].mean(), to_replace=0)
# SkinThickness and Insulin should not have the zero values,
# but many are missing, so we skip at this step
```

<table>
<tr><th>Column</th><th>Before filling zeros</th><th>After filling zeros</th></tr>
<tr><td>Glucose</td><td>
</td><td>
</td></tr>
<tr><td>BloodPressure</td><td>
</td><td>
</td></tr>
<tr><td>BMI</td><td>
</td><td>
</td></tr>
</table>

5.1.2. Fill by Regression Method

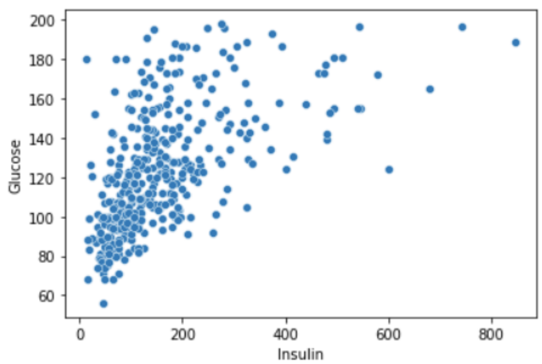
The regression method is a very useful when the columns that having zero-values are very correlated with the other columns. So we will try this method on these columns: **Insulin, SkinThickness.**

Correlation Matrix, **Pearson metric** *(on dataset that have Insulin != 0 and SkinThickness != 0):*

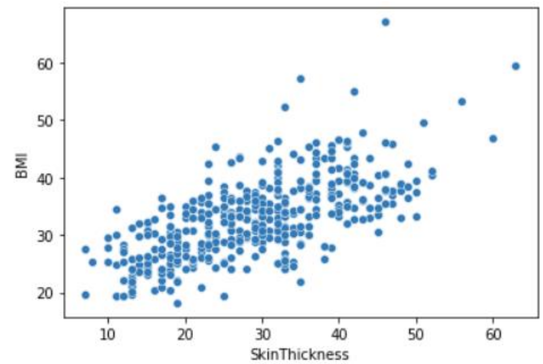


We can see that there are some high correlations between:

Insulin and Glucose (0.58).

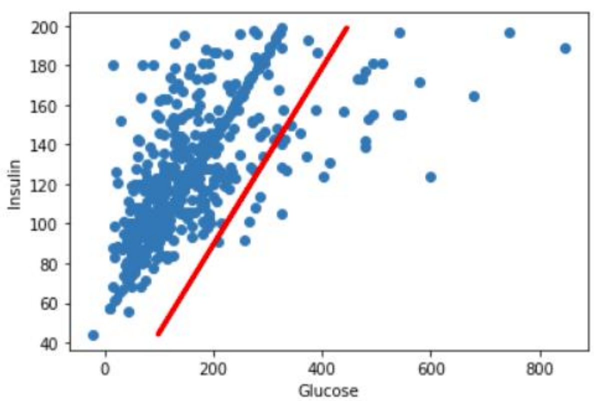


SkinThickness and BMI (0.66).

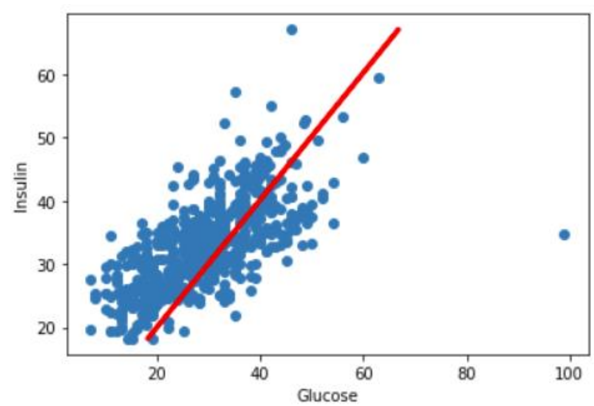


We try to estimate the zero-values by Linear Regression:

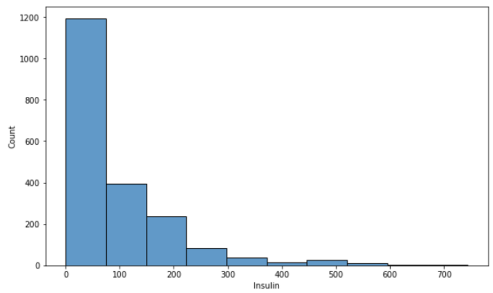
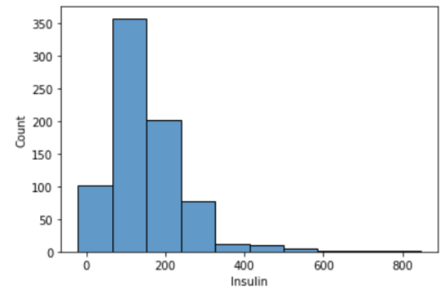
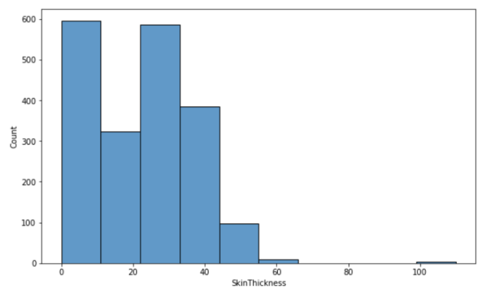
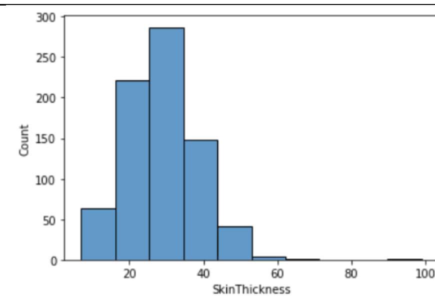
Insulin = Glucose * 2.23952761



SkinThickness = BMI * 0.99549695



Now the filling result:

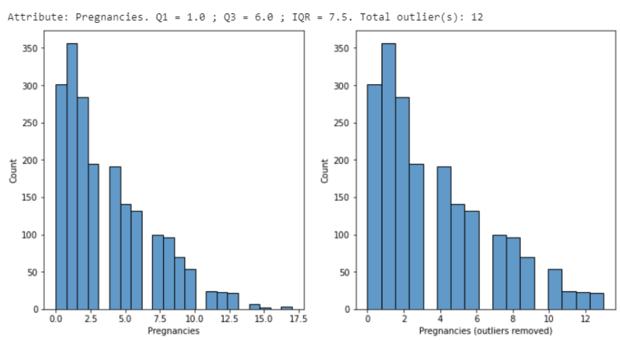
Column	Before filling zeros	After filling zeros
Insulin		
SkinThickness		

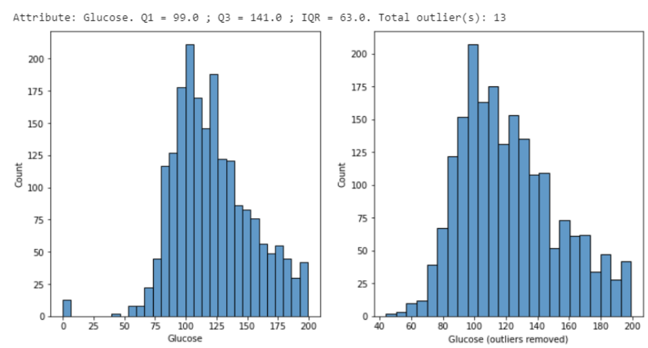
5.2. Outliers Removal

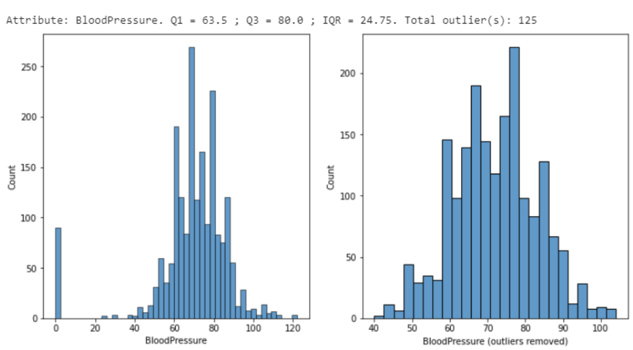
We would consider outliers by using **Inter-quartile Range** method. In case the data point is out of range

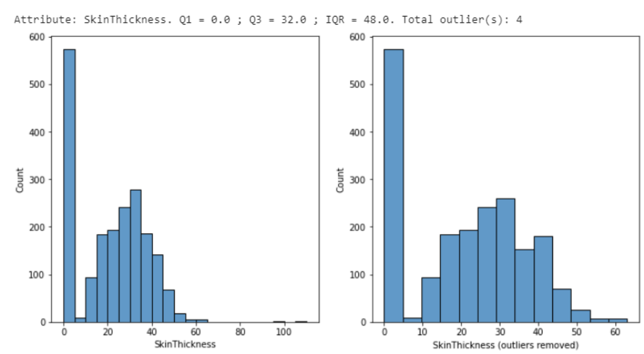
$$[\ Q1 - 1.5*IQR,\ Q3 + 1.5*IQR\ ]$$

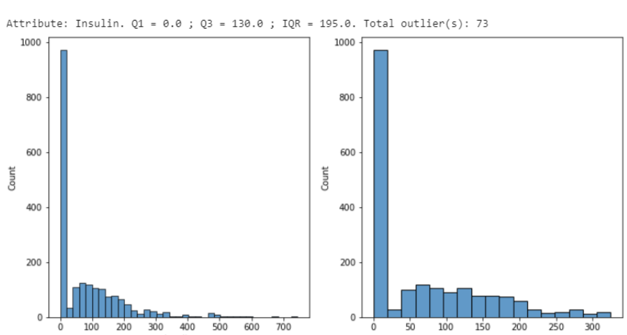
it might be **outliers**.

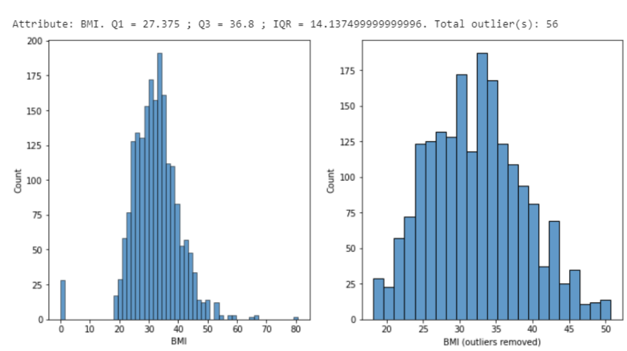


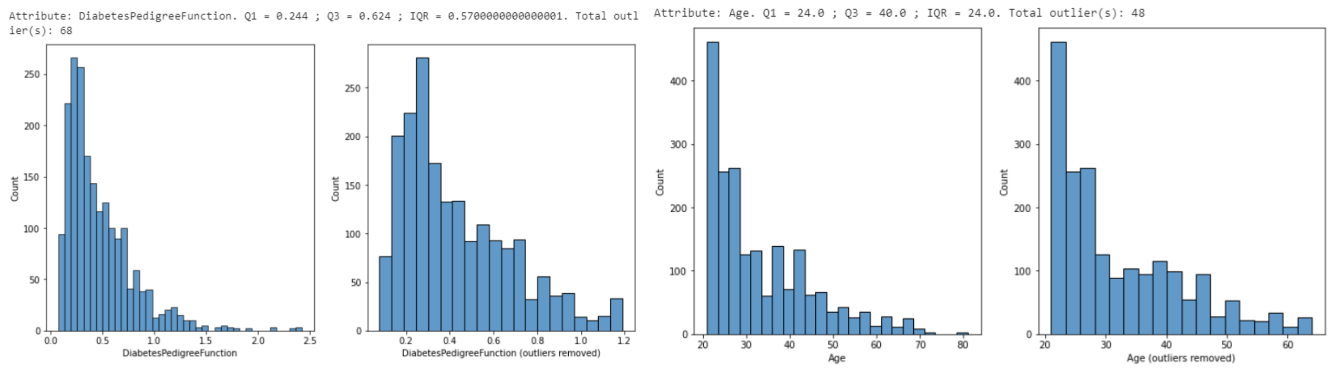












Total removed outliers in dataset: 11,2%

6 Solution

In this section, I will do some different classification methods to classify if the people have diabetes or not, based on their medical details.

Before we do the classification tasks, we need to normalize the input first.

6.1. Normalization

We standardize features by removing the mean, then scaling dataset to unit variance.

Note that we will not standardize the **Outcome** column, as it is a **categorical column**.

The standardized score is calculated as below:

$$Z = (X - U) / S$$

Where:

- X: The value of sample in dataset.
- U: The mean of dataset.
- S: The standard deviation of dataset.

```
df = data.drop(['Outcome'], axis=1)
```

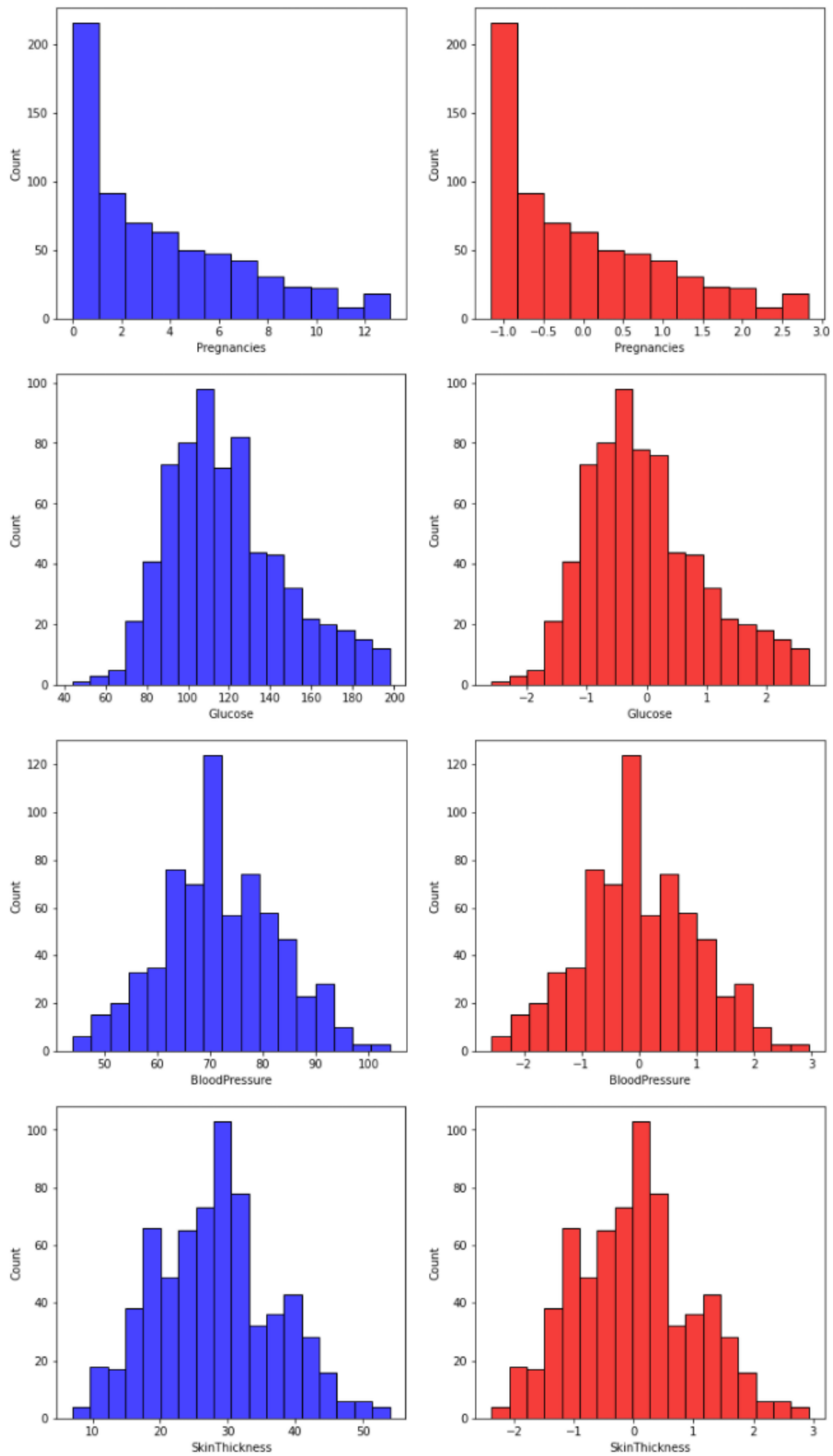
```
df.mean()
```

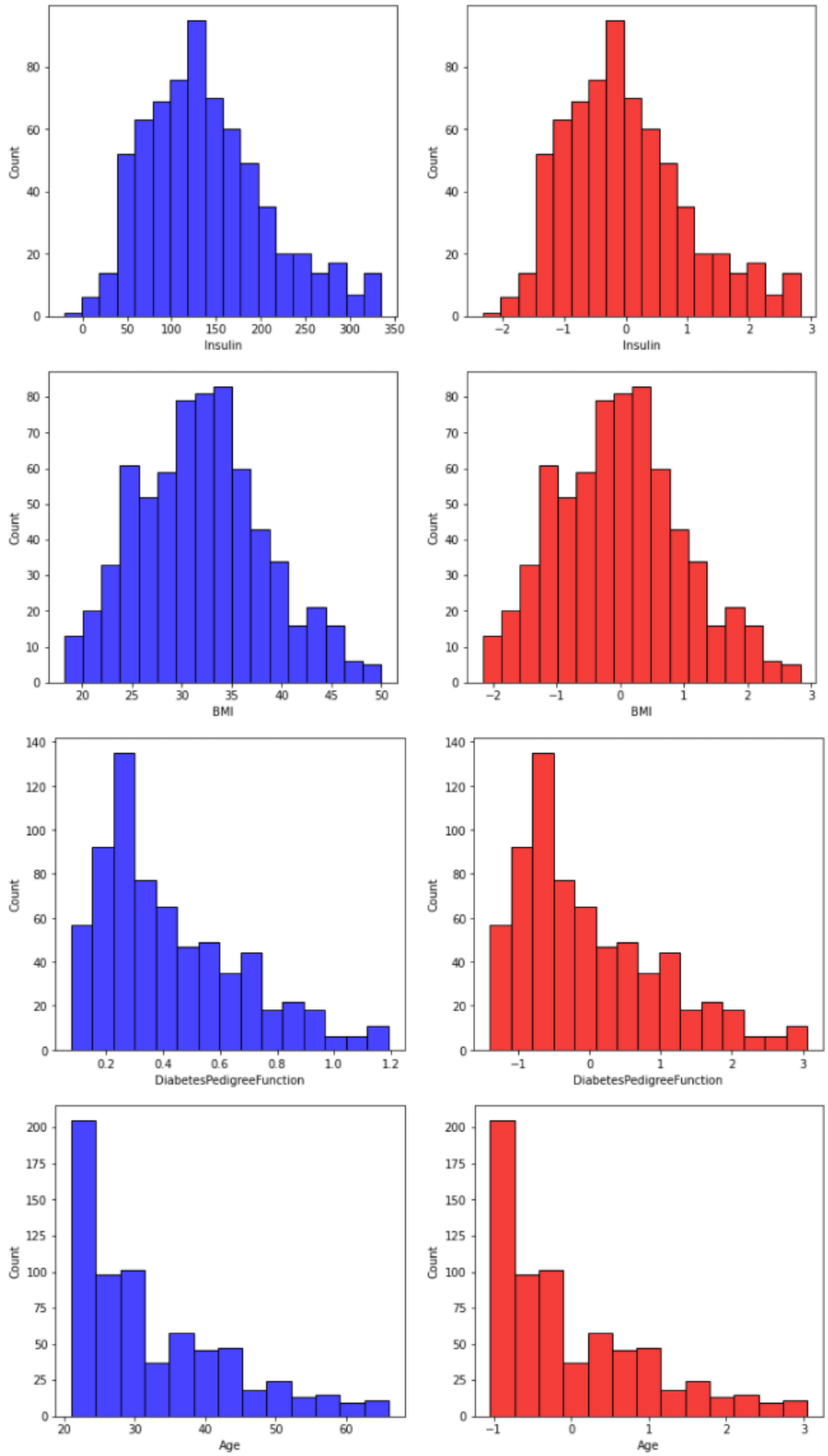
```
Pregnancies                   3.70350
Glucose                     121.18250
BloodPressure                69.14550
SkinThickness                20.93500
Insulin                      80.25400
BMI                          32.19300
DiabetesPedigreeFunction      0.47093
Age                          33.09050
dtype: float64
```

```
df.std()
```

```
Pregnancies                   3.306063
Glucose                      32.068636
BloodPressure                19.188315
SkinThickness                16.103243
Insulin                     111.180534
BMI                           8.149901
DiabetesPedigreeFunction      0.323553
Age                          11.786423
dtype: float64
```

The result after normalization *(the blue shows original, the red shows normalized)*


Count
Pregnancies
Count
Pregnancies
Count
Glucose
Count
Glucose
Count
BloodPressure
Count
BloodPressure
Count
SkinThickness
Count
SkinThickness


Count
Insulin
Count
Insulin
Count
BMI
Count
BMI
Count
DiabetesPedigreeFunction
Count
DiabetesPedigreeFunction
Count
Age
Count
Age

6.2. Model Training

In this project, I have done on some different kind of classification methods as below:

- Logistic Regression

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

- Decision Tree

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier

- Random Forest

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

- SVM (Linear Kernel)

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

- kNN

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

The implementation has been done by scikit-learn library (Python), so I just use the model in the library.

The data is split to **trainset** and **testset**, with ratio **7/3**: 70% data for training and 30% for testing, respectively.

The splitting data and training code:

```
# We need to split dataset to train/test data
# 70% train, 30% test
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=0.7)
```

```
# We will do some different type of algorithms to determine which is the best
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier

list_models = [
    LogisticRegression(),
    DecisionTreeClassifier(),
    RandomForestClassifier(),
    SVC(),
    KNeighborsClassifier(),
]
```

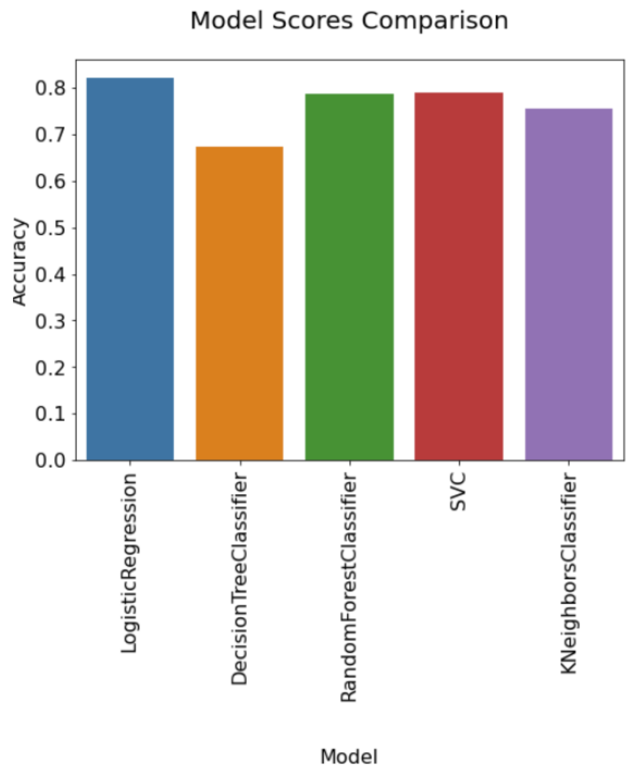
```
# Now training
for model in list_models:
    model.fit(x_train.values, y_train)
```

6.3. Result

The testing result are calculated on 30% data (testset).

6.3.1. Accuracy Score

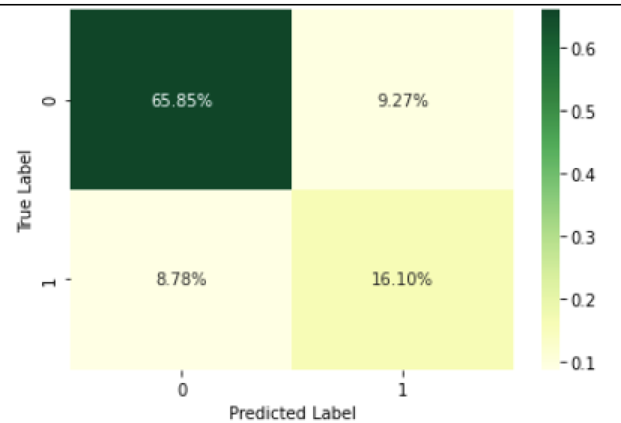
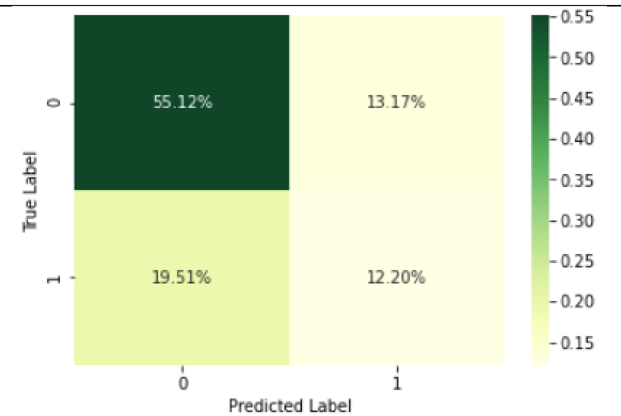
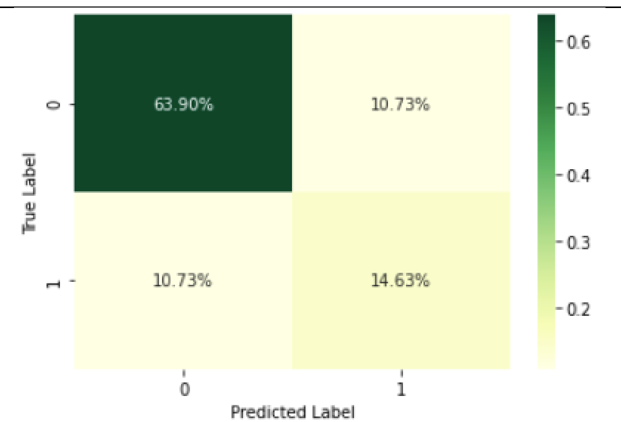
model_name	accuracy_score
LogisticRegression	0.819512
DecisionTreeClassifier	0.673171
RandomForestClassifier	0.785366
SVC	0.790244
KNeighborsClassifier	0.756098

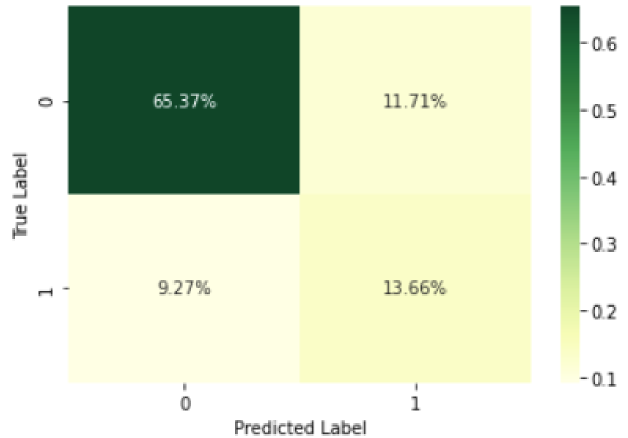
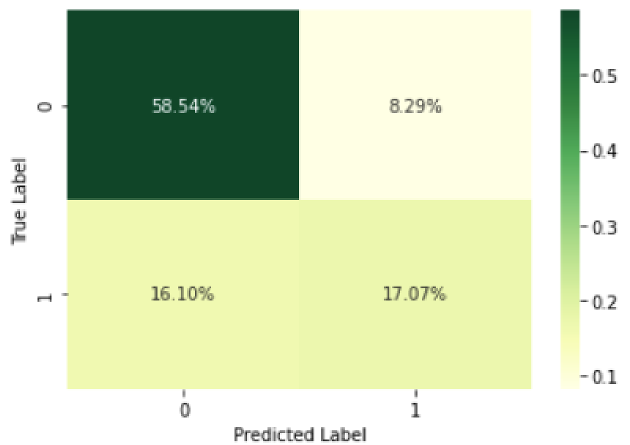


With accuracy metrics, the **LogisticRegression** gave the best result.

6.3.1. Confusion Matrix

To analyze more in detail on the prediction of classification problems, we use confusion matrix.

Model	Confusion Matrix
LogisticRegression	
DecisionTreeClassifier	
RandomForestClassifier	

SVM	
kNN	

This classification is done on **medical examination**, which is the False Negative is one of the most importance metrics. Based on this result, Logistic Regression method gave the best result with **8.78% (lower is better).**

7 Conclusion

In this project, I have done end-to-end flow to analyze, process and do some classification methods on the Diabetes Prediction task.

The precision, recall and f-score of all methods are below:

model_name	precision	recall	fscore
LogisticRegression	0.758484	0.761841	0.760128
DecisionTreeClassifier	0.609666	0.595879	0.599341
RandomForestClassifier	0.716566	0.716566	0.716566
SVC	0.707139	0.721923	0.713696
KNeighborsClassifier	0.728695	0.695309	0.705460

In conclusion, **the simple Logistic Regression** performs the **best result** on the provided dataset.

This result might be improved if we use other complex methods, e.g., Deep Learning.