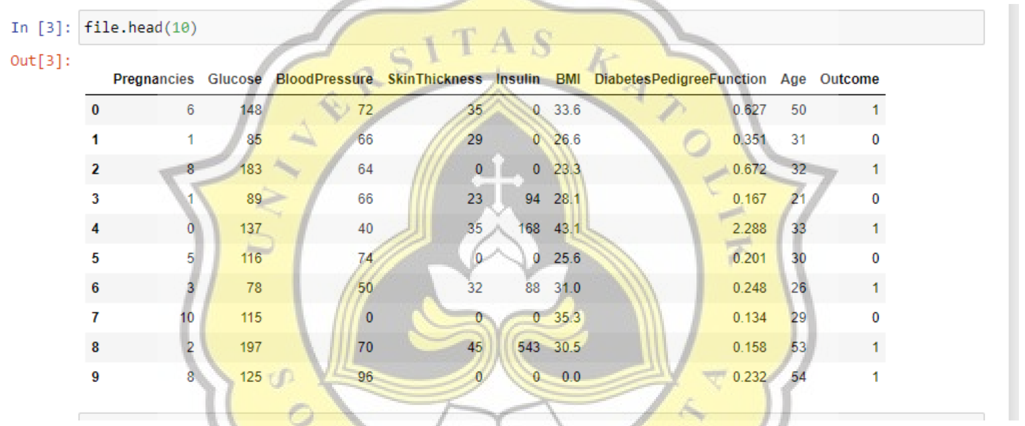
CHAPTER 5
IMPLEMENTATION AND TESTING

5.1 Implementation

This chapter is talking about how the code being implemented after the long explanation in the previous chapter. Before jump to the data preparation, first the data must be known and checked by the user. Since the data has a lot of items, just show the top 10 of the data as the image below.



```
In [3]: file.head(10)
Out[3]:
```

	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
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1

***Illustration** 5.1. The head of the dataset*

To do the data preparation, the heatmap from the dataset must be shown with the code:

```
1 plt.figure(figsize=(20,20))
2 sns.heatmap(file[:-1].corr(), annot=True,cbar=False)
3 plt.show()
```

From line 1, give the output size of the image is 20 x 20. Line 2 means to call the heatmap with the column, without the last column (written: -1), with the correlation between the data. And line 3 shows the heatmap as below.

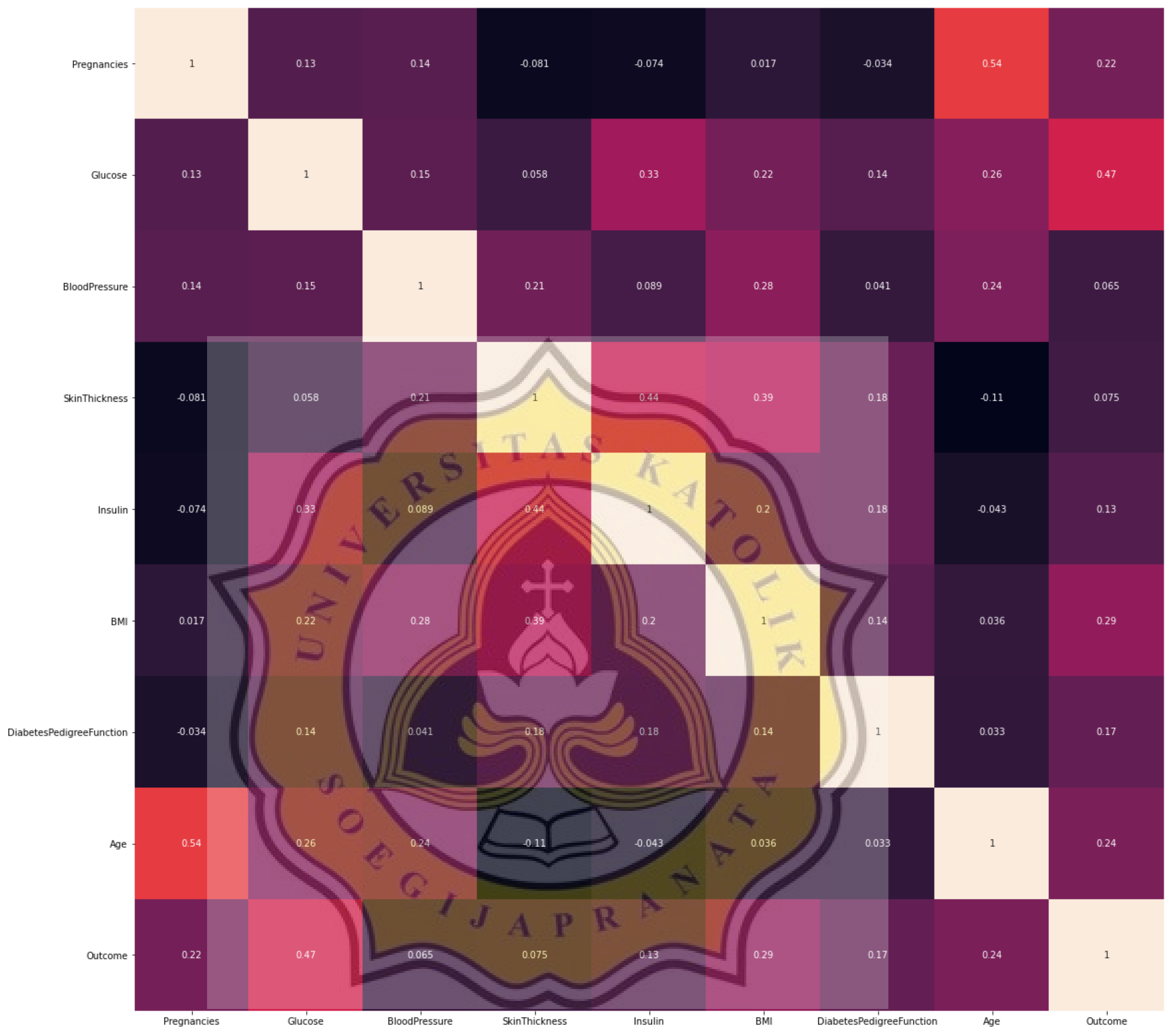


__Illustration 5.2__. Heatmap from the dataset

Each square in the heatmap shows the correlation between the variables on each axis, the correlation ranges from -1 to +1. The interpretation of the heatmap is the values closer to zero mean there is no linear trend between the two variables, the closer to 1 the correlation is the more positively correlated they are, and closer to -1 is similar, but instead of both increasing one variable will decrease as the other increases. After taking a look at the correlation in the heatmap, the next step is to show the histogram between each variable and their density. The main concern of this step is to find the normal distribution of each variable. Normal distribution is a statistical function that describes the likelihood of obtaining the most possible values that a real-valued random variable can take. The Normal distribution has a general form with formula:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

Where, μ = Mean, σ= Standard deviation, x = input value.

In this project, the histogram will show the probability of every variable to the correlation with the prediction result. From the picture, the shape of the histogram will show which variables has a higher correlation. There will be 8 histograms shown below.

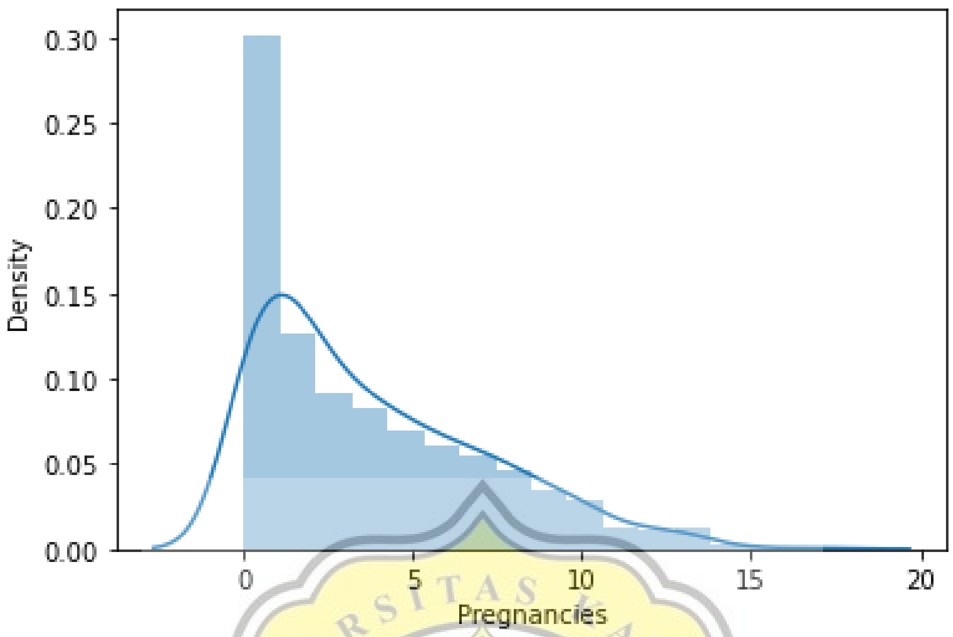


***Illustration** 5.3. Pregnancy and it's density graph*

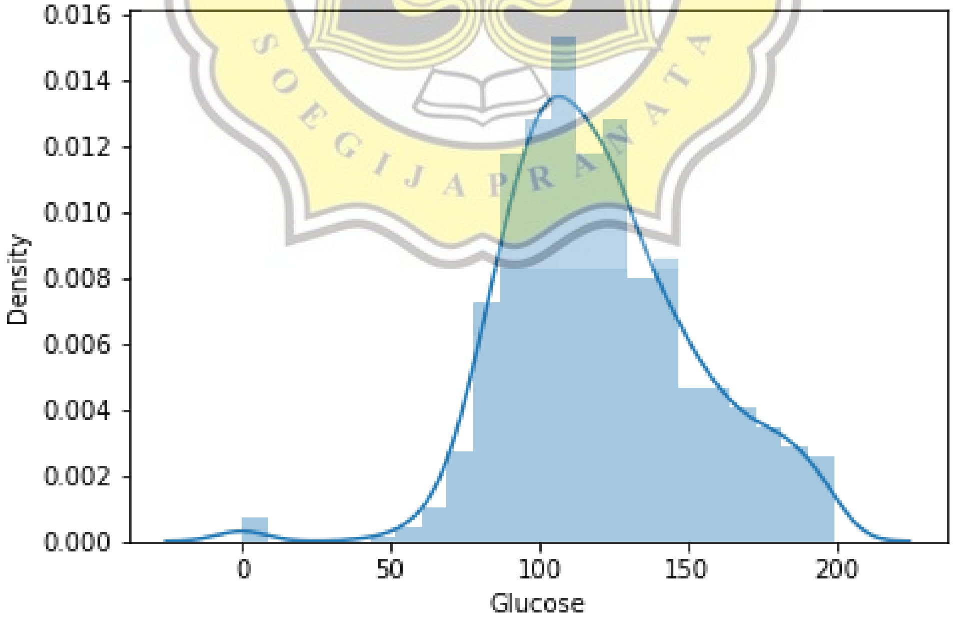


***Illustration 5.4.** Glucose and it's density graph*

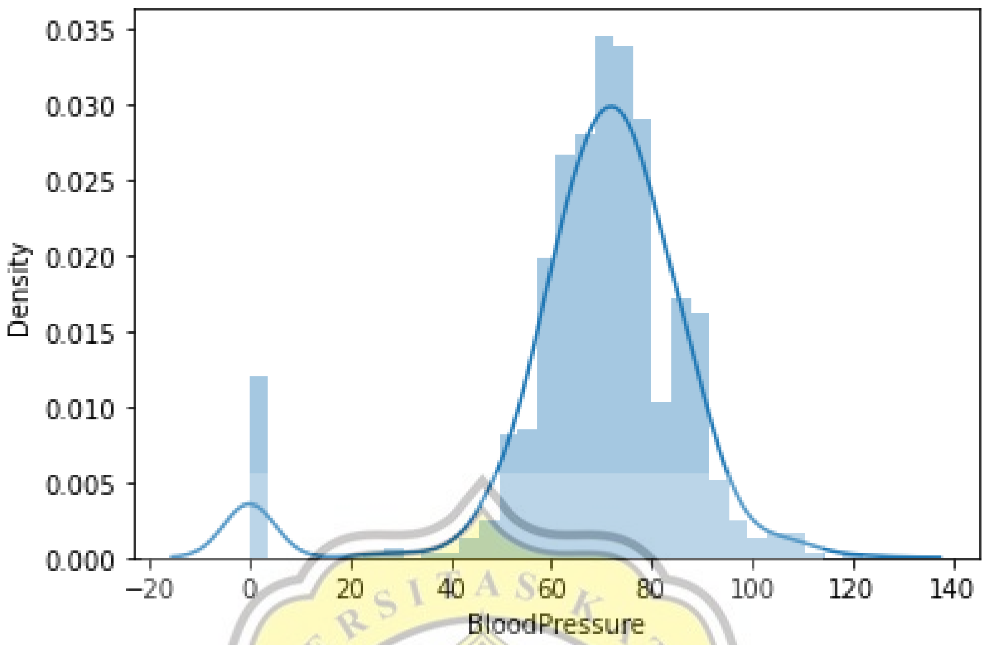


***Illustration** 5.5. Blood pressure and it's density graph*

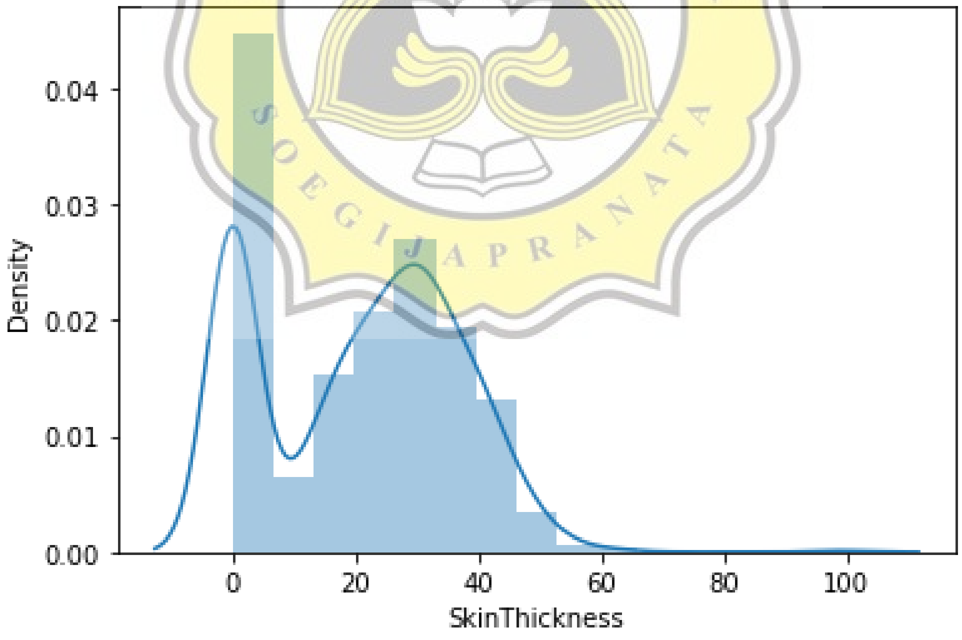


***Illustration 5.6.** Skin thickness and it's density graph*

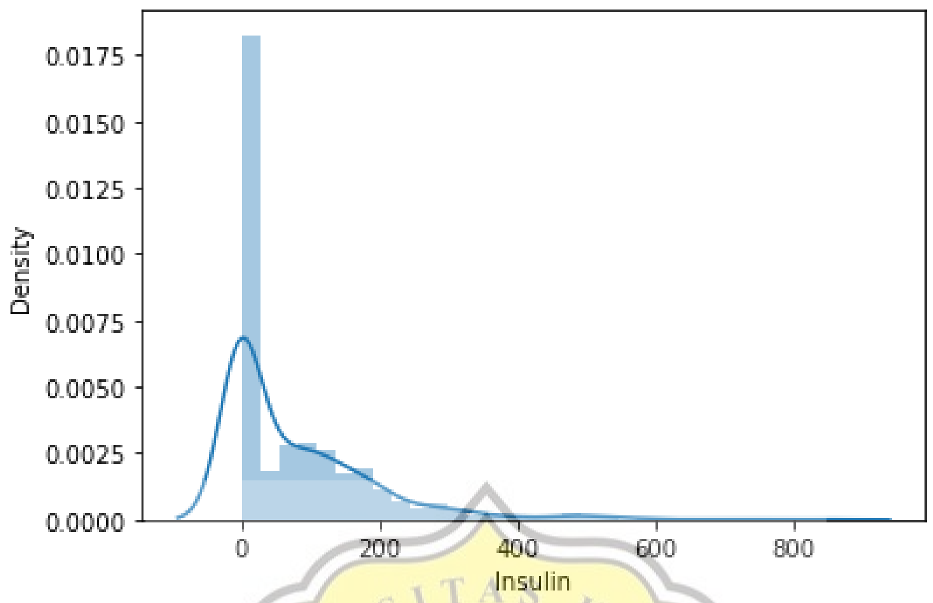


***Illustration** 5.7. Insulin and it's density graph*

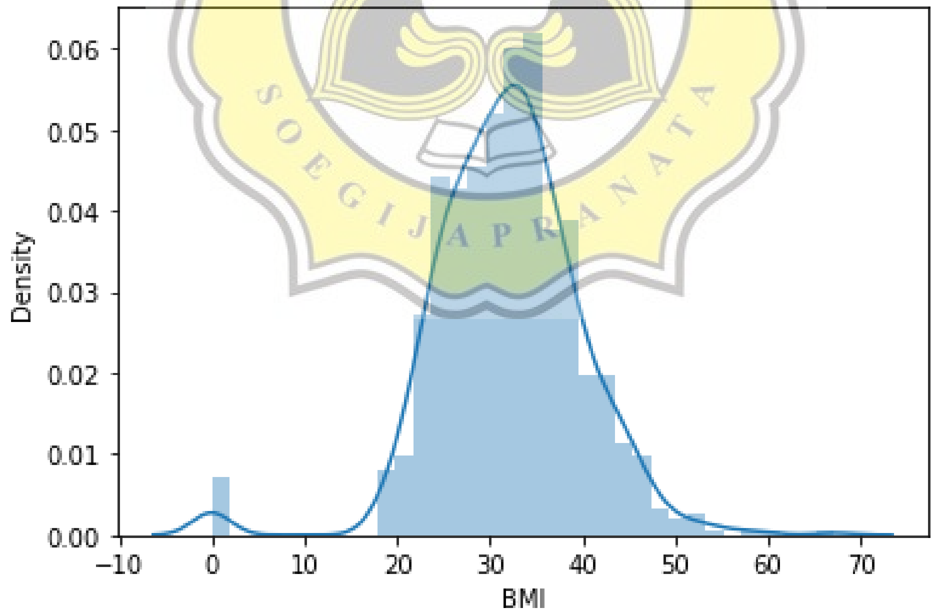


***Illustration 5.8.** BMI and it's density graph*

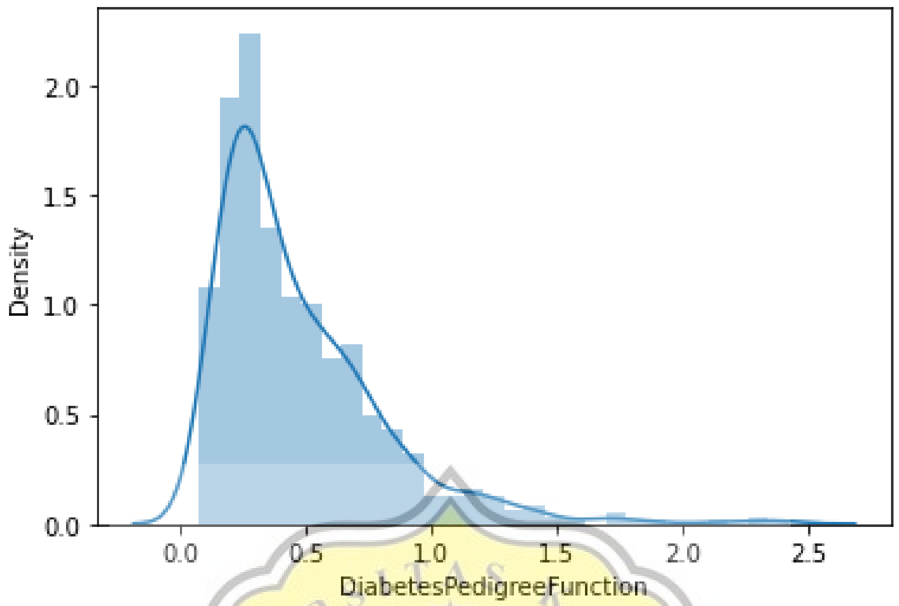


Illustration 5.9. *Diabetes pedigree function and it's density graph*

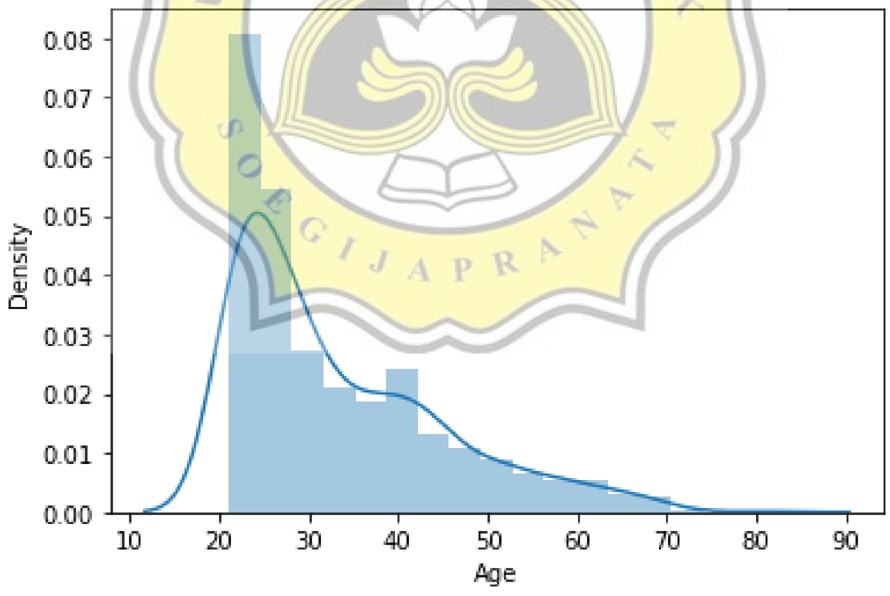


Illustration 5.10. *Age and it's density graph*

The histograms above shows that the variable skin thickness and blood pressure have an unusual shape. The shape of their graphs has two peak that will affect the prediction result if they are used. To avoid that, feature transformation will be used, so the unusual histograms will return into usual with change the unknown values of the variables. The code of the feature transformation will be shown below.

```
1  file['BMI'] = np.where(file['BMI'] == 0, np.nan, file['BMI'])
2  file['Glucose'] = np.where(file['Glucose'] == 0, np.nan, file['Glucose'])
3  file['BloodPressure'] = np.where(file['BloodPressure'] == 0, np.nan,
4  file['BloodPressure'])
5  file['SkinThickness'] = np.where(file['SkinThickness'] == 0, np.nan,
6  file['SkinThickness'])
7
8  file['BMI'].fillna(27, inplace = True)
9  file['Glucose'].fillna(file['Glucose'].mean(), inplace = True)
10 file['BloodPressure'].fillna(file['BloodPressure'].mean(), inplace = True)
11 file['SkinThickness'].fillna(file['SkinThickness'].mean(), inplace = True)
```

In line 1-6 is code to find the 0 value or the unknown value of each variable that written (BMI, glucose, blood pressure, and skin thickness). In line 8-11 is code to replace the unknown value into the mean of each variable data. After the code has been run, it will make a huge change to the histogram. The new histogram will be shown below.

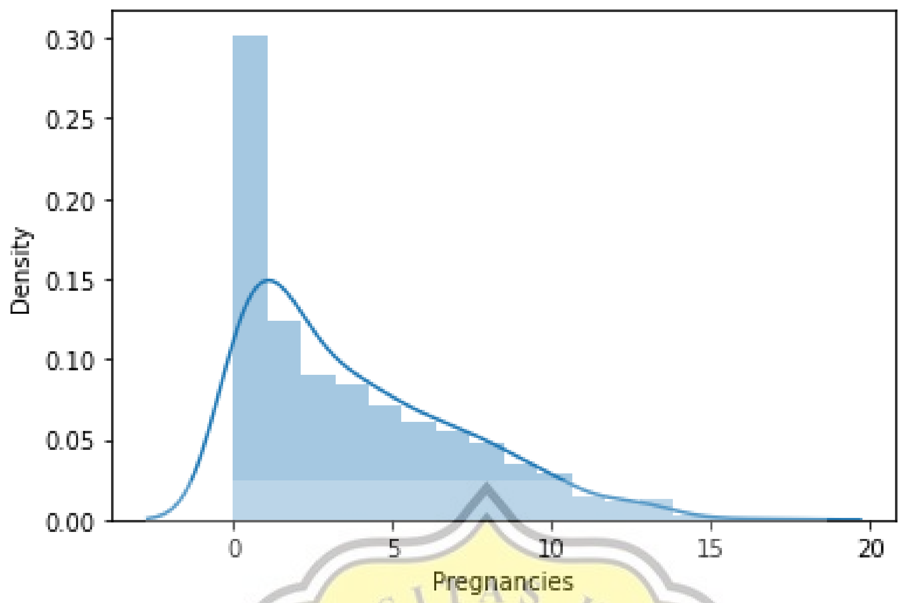


***Illustration** 5.11. The new histogram of pregnancies*

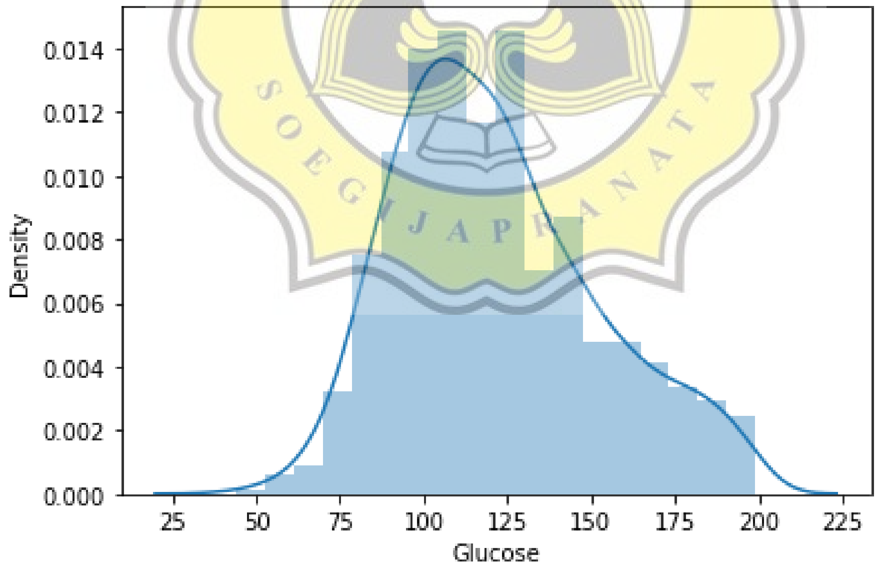


***Illustration 5.12.** The new histogram of glucose*

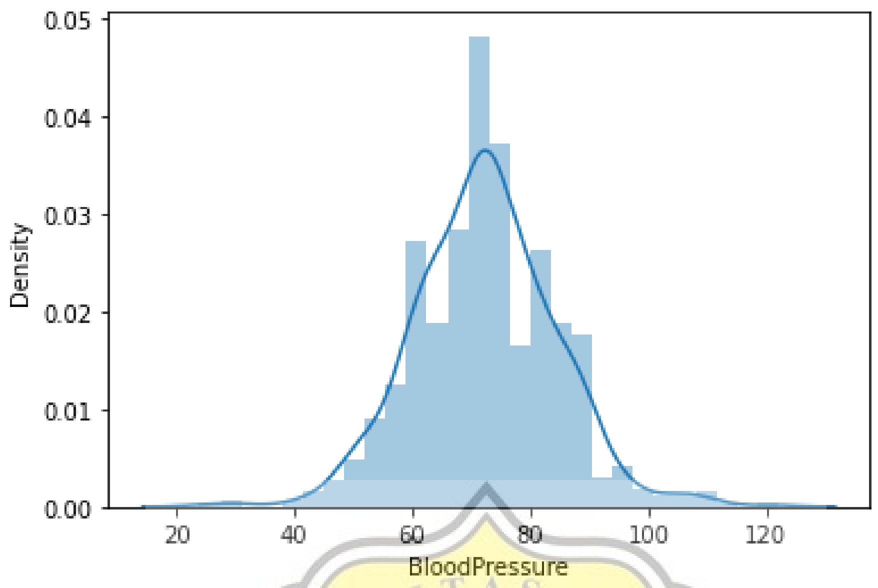


***Illustration** 5.13. The new histogram of Blood Pressure.*

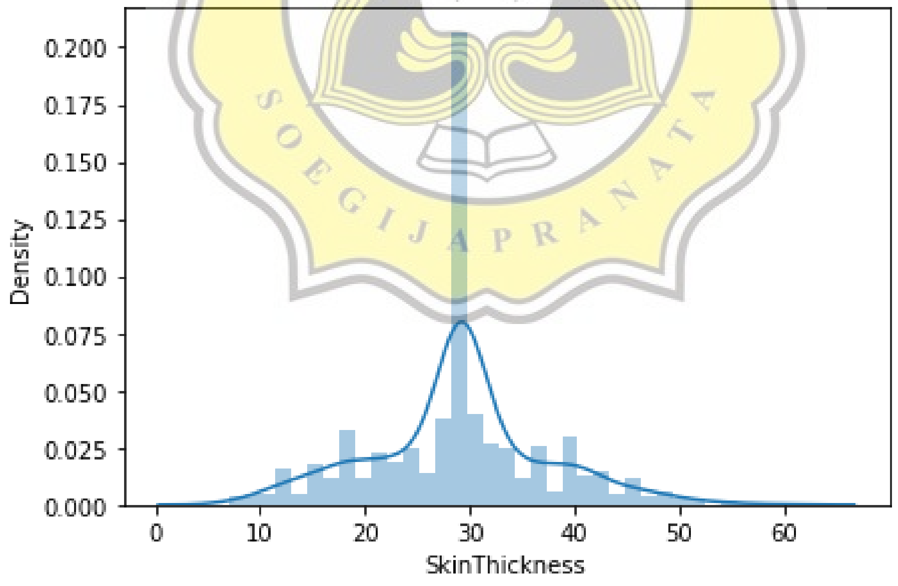


***Illustration 5.14.** The new histogram of Skin Thickness.*

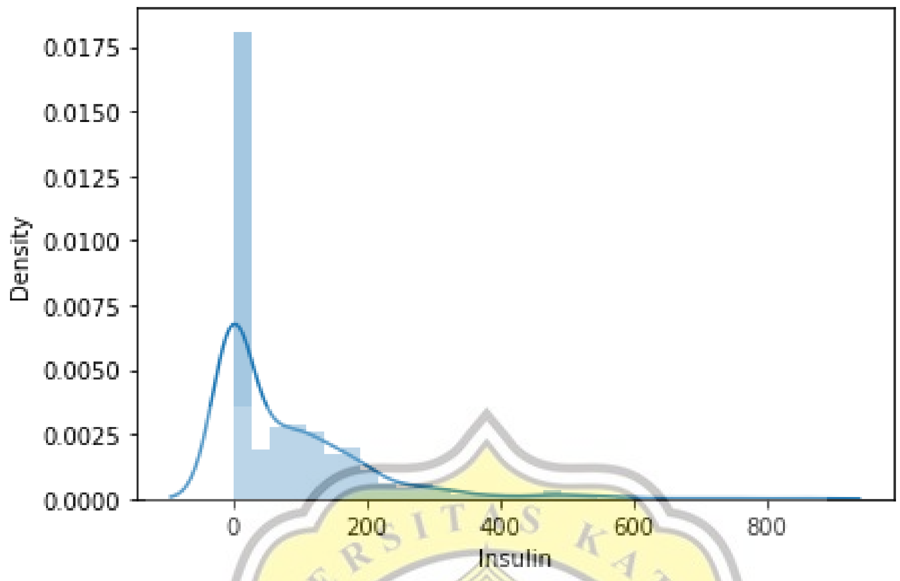


Illustration 5.15. *The new histogram of insulin.*

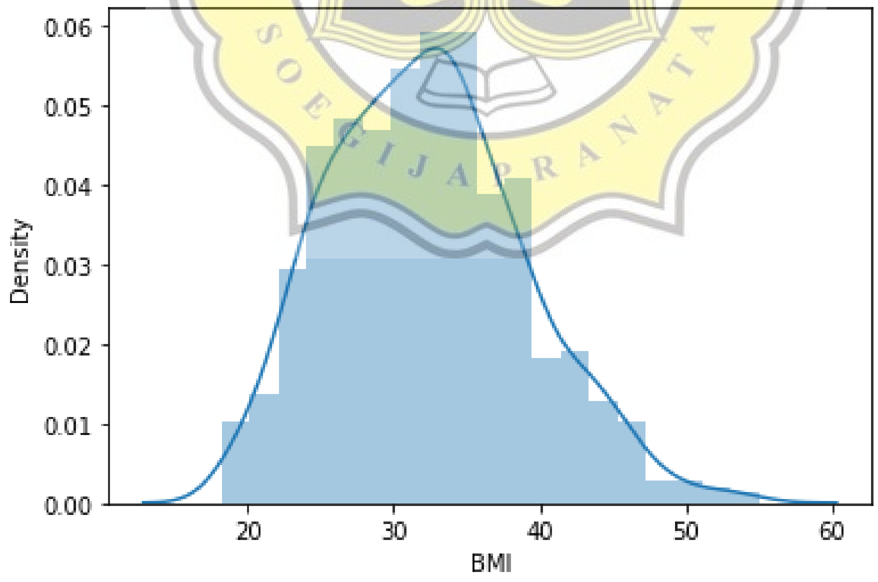


Illustration 5.16. *The new histogram of BMI.*

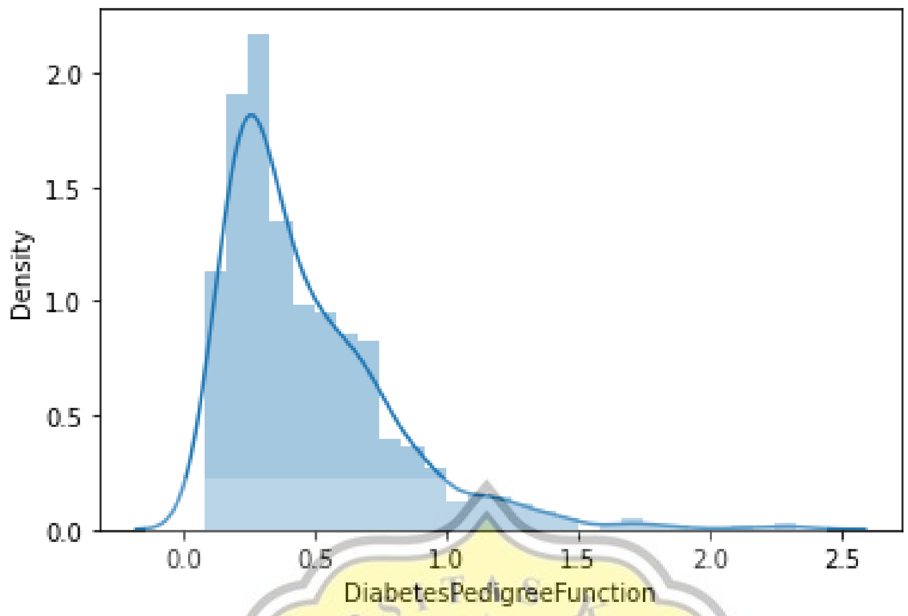


***Illustration** 5.17. The new histogram of Diabetes Pedigree Function.*

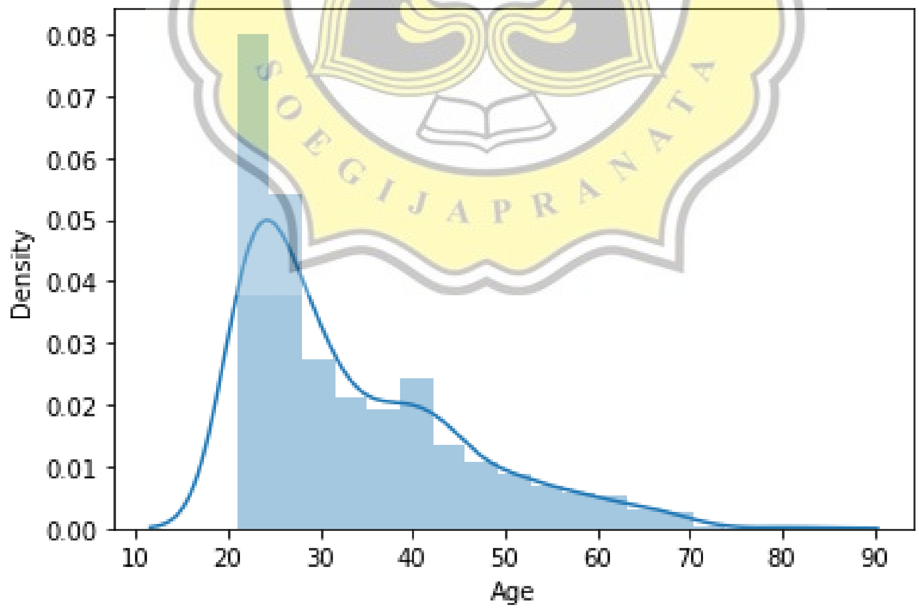
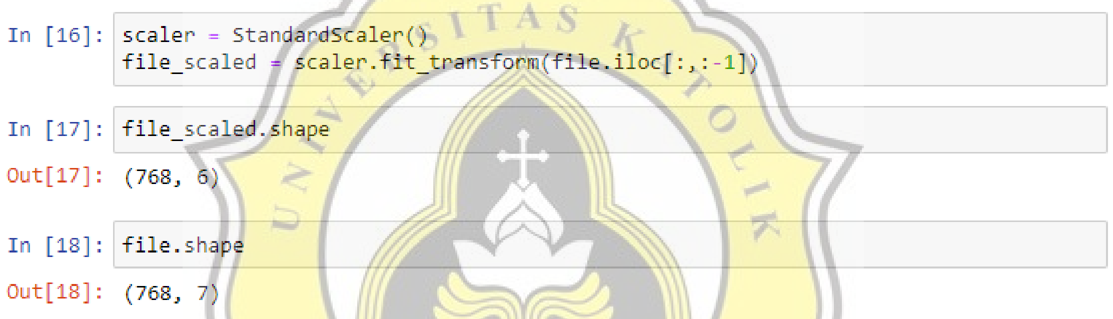


***Illustration 5.18.** The new histogram of Age.*

From the whole images above, the histograms now show all normal distribution histograms with the usual data. These new histogram shows that all the data now could be processed and the result will be optimal since there is no unknown data.

After transform the unknown value in the variables, then the data from all variables will be scaled in range 0 and 1. This step will make the data easier to be read by the computer. Since the last column of the data is an outcome (the patient has diabetes or not), then the last column will be deleted. The image below will show the result of the cleaning and scaling.



```
In [16]: scaler = StandardScaler()
         file_scaled = scaler.fit_transform(file.iloc[:,:-1])

In [17]: file_scaled.shape

Out[17]: (768, 6)

In [18]: file.shape

Out[18]: (768, 7)
```

***Illustration** 5.19. Cleaning and scaling the data*

When the data has been trained, the data will be tested split to prevent the model from overfitting, underfitting, and to accurately evaluate the model. To evaluate a model's predictive performance, the same data used for training cannot be used. It is because unbiased evaluation is needed to properly use the measures, assess the model's predictive performance, and validate the model. To do that, fresh data that has not been seen by the model before is needed by splitting the data set before it is used. In this project, the data split into 10% of the data that will be tested as the code below.

```
1  # from sklearn.model_selection import train_test_split
2
3  def train_valid(data, frac= 0.1):
4      data_x = data
5      data_y = file['Outcome']
6
7      train_x, valid_x, train_y, valid_y =
8  train_test_split(data_x,data_y,test_size=frac)
9      return train_x,valid_x,train_y,valid_y
10 train_x, valid_x, train_y, valid_y = train_valid(file_scaled)
```

In line 1, the train test split from the Sci-Kit learn library is imported. Code in lines 3-9 used to define train valid data using the 10% of the data, it shows by frac of the data is 0.1. This line also defines the variables that are used to train test split with the return of the data that has been defined before.

To make sure that the outcome data from the dataset is a good data that contain 0 and 1, data visualization is needed. In this case, seaborn library is used to count the plot of the "outcome" data and the result is shown on the image below.

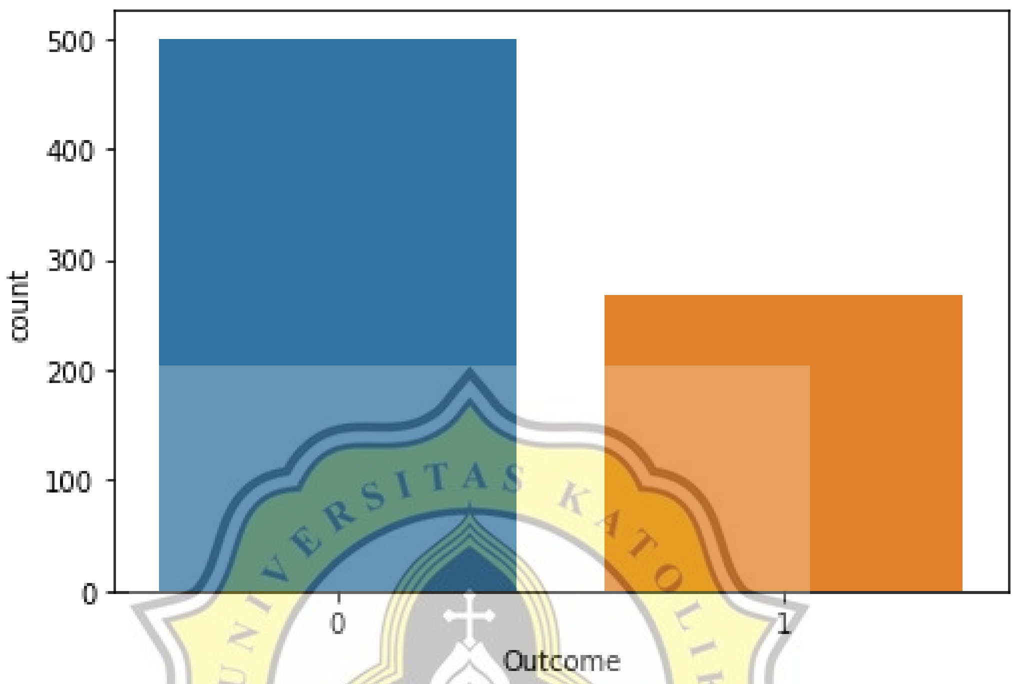


Illustration 5.20. *The count plot of 'outcome' data.*

From the graph above, the data contains 0 and 1 as the variable that affect whether the patient has a diabetes or not. The sum of the outcome show that it is a healthy and usual data and it can be processed to the next step.

After doing some data preprocessing, the next step is to show the things that matter to the data and to optimize the model.

```
1  import time
2  from sklearn.metrics import accuracy_score, roc_auc_score,
3  plot_confusion_matrix, roc_curve, classification_report
4  def run_model(model, X_train, y_train, X_test, y_test, verbose=True):
5      t0=time.time()
6      if verbose == False:
7          model.fit(X_train,y_train, verbose=0)
8      else:
9          model.fit(X_train,y_train)
10     y_pred = model.predict(X_test)
11     accuracy = accuracy_score(y_test, y_pred)
12     roc_auc = roc_auc_score(y_test, y_pred)
13     time_taken = time.time()-t0
14     print("Accuracy = {}".format(accuracy))
15     print("ROC Area under Curve = {}".format(roc_auc))
16     print("Time taken = {}".format(time_taken))
17     print(classification_report(y_test,y_pred,digits=5))
18
19     probs = model.predict_proba(X_test)
20     probs = probs[:, 1]
21     fper, tper, thresholds = roc_curve(y_test, probs)
22     plot_roc_cur(fper, tper)
23
24     plot_confusion_matrix(model, X_test, y_test,cmap=plt.cm.Blues,
25 normalize = 'all')
26
27     return model, accuracy
```

From line 1, import time is used to show the time used when training the data. The second line shows the way to import library from Sci-kit learn metrics to show accuracy score, etc. as it showed. Line 4-27 is to defines the run model to train the dataset. The main concern of this class is to show the accuracy, f-1 score, precision, recall, ROC curves, and confusion metrics from the data. Accuracy is the ratio of the number of correct predictions to the total number of input samples, with the formula:

$$Accuracy = \frac{Number\ of\ corrected\ predictions}{Total\ number\ of\ predictions\ made}$$

To give the best output, evaluating the performance of a classification model is not stop in accuracy, but more. Precision, to show the proportion of relevant research (correctly predicted yes) in the list of all returned search result (total predicted yes). Recall, to show the ratio of the relevant results (correctly predicted yes) returned by the search engine to the total number of the relevant result could have ben returned (total actual yes). F1-score to show when there are imbalanced classes with the formula:

$$F1 = \frac{2\,(precision)(recall)}{precision + recall}$$

With a note that the precision value and the recall value need to be large, and it will be the highest when P&R are equals to one.

ROC (Receiver Operating Characteristic) curves used to visualize the performance of the binary classifier, the meaning classifier with two possible output classes, it plots two parameters: true positive rate/recall (TPR) with the formula:

$$TPR = \frac{TP}{TP + FN}$$

and the second one false positive rate (FPR) with formula:

$$FPR = \frac{FP}{FP + TN}$$

Where the TP is true positive, TN is true negative, FP is false positive and FN is False negative. ROC curves can be extended to problems with three or more class, and it is very useful even if the predicted probabilities are nor properly calibrated.

And the confusion matrix is used to describe the performance of the classification model. This step was an important step to know the model evaluation to begin the logistic regression. After that, implement the Naïve Bayes using Sci-Kit learn library as the code below:

```
1 from sklearn.naive_bayes import GaussianNB
2 NB =GaussianNB()
3 NB, accuracy_NB = run_model(NB,train_x,train_y,valid_x,valid_y)
```

In line 1, the Naïve Bayes algorithm was called from the library, then in line 2, the Naïve Bayes was declared into NB variable. In line 3, to get the Naïve Bayes and the accuracy of the Naïve Bayes, the model is running with the parameters as it shows in the code.

The Gaussian Naïve Bayes from the scikit library is done, move to logistic regression without a library, the code is below:

```
1  class LogisticRegression:
2      def __init__(self,lr=0.01,epochs=1000):
3          self.lr = lr
4          self.epochs = epochs
5          self.weights = None
6
7      def sigmoid(self,z):
8          return 1/(1+ np.e**(-z))
9
10     def cost_function(self,X,y):
11         z = np.dot(X,self.weights)
12         predict1 = y*np.log(self.sigmoid(z))
13         predict0 = (1 - y) * np.log(1 - self.sigmoid(z))
14         return -sum(predict1 + predict0) / len(X)
15
16     def fit(self,X,y):
17         loss = []
18         self.weights = np.random.rand(X.shape[1])
19         n =len(X)
20
21         for i in range(self.epochs):
22             yhat = self.sigmoid(np.dot(X,self.weights))
23             self.weights -= self.lr*np.dot(X.T,yhat-y)/n
24             loss.append(self.cost_function(X,y))
25
26         self.weights = self.weights
27         self.loss = loss
28
29     def predict(self,X):
30         z = np.dot(X, self.weights)
31
32         return [1 if i > 0.5 else 0 for i in self.sigmoid(z)]
33
34 def model_accuracy(preds,y_true):
35     ret = (preds == y_true)
36     return sum(ret)/len(preds)
```

In class logistic regression there are some steps, in line 2-5 is used to make the variable global to declare, LR, the number of epochs, and declare the weights. The lines 7&8 show to define the sigmoid with the return value is decided 0 or 1.

Line 10-14 is to define the cost function using dot product between the matrix X and current weight, with the returned value is set not negative. Line 16-19 is to train the data using the old weight and random numbers, from line 21-24 performs the gradient descent to know the optimum gradient as explained in the previous chapter. Then, in line 26 & 27 is to renew the variables. In line 29, it shows the code to predict using the optimal data that got from the previous code. Then the last line 34-36 is to check the accuracy of the models just like in the scikit library before.

To know this model is work efficiently, the mean squared error must be descending every iteration, so the coding below is to check the MSE.

```
1 xs = np.arange(len(model.loss))
2 ys = model.loss
3
4 plt.plot(xs, ys, lw=3, c='#087E8B')
5 plt.title('Loss per iteration (MSE)', size=20)
6 plt.xlabel('Iteration', size=14)
7 plt.ylabel('Loss', size=14)
8 plt.show()
```

Line 1&2 is to make a graph to show the model loss per iteration, line 4-8 is to set what the graph will conclude: Loss per iteration (define by Mean Squared Error/MSE), The sum of iteration that has been done, and Loss per iteration.

The next step is to find the best learning rate by seeing loss in the model using different learning rate. Learning rate that used as sample here is 0.5, 0.1, 0.01, 0.001, the code as follows:

```
1.losses = {}
2.for lr in [0.5, 0.1, 0.01, 0.001]:
3.    model = LogisticRegression(lr = lr)
4.    model.fit(train_x, train_y)
5.    losses[f'LR={str(lr)}'] = model.loss
6.
7.
8.xs = np.arange(len(model.loss))
9.
10.plt.plot(xs, losses['LR=0.5'], lw=3, label=f"LR = 0.5, Final =
11.{losses['LR=0.5'][-1]:.2f}")
12.plt.plot(xs, losses['LR=0.1'], lw=3, label=f"LR = 0.1, Final =
13.{losses['LR=0.1'][-1]:.2f}")
14.plt.plot(xs, losses['LR=0.01'], lw=3, label=f"LR = 0.01, Final =
15.{losses['LR=0.01'][-1]:.2f}")
16.plt.plot(xs, losses['LR=0.001'], lw=3, label=f"LR = 0.001, Final =
17.{losses['LR=0.001'][-1]:.2f}")
18.plt.title('Loss per iteration (MSE) for different learning rates',
19.size=20)
20.plt.xlabel('Iteration', size=14)
21.plt.ylabel('Loss', size=14)
22.plt.legend()
23.plt.show()
```

Line 1-5 is to the best learning rate using the MSE or loss per iteration, this line is showing the code to do looping between different learning rate. After that, line 10-23 is to make the MSE graph with the different learning rate, the best learning rate will be used is the learning rate with the MSE descending.

5.2 Testing

This chapter will show that the code was run successfully and the output like what its expected. From the first Naïve Bayes algorithm with the library returned the output as the image below.

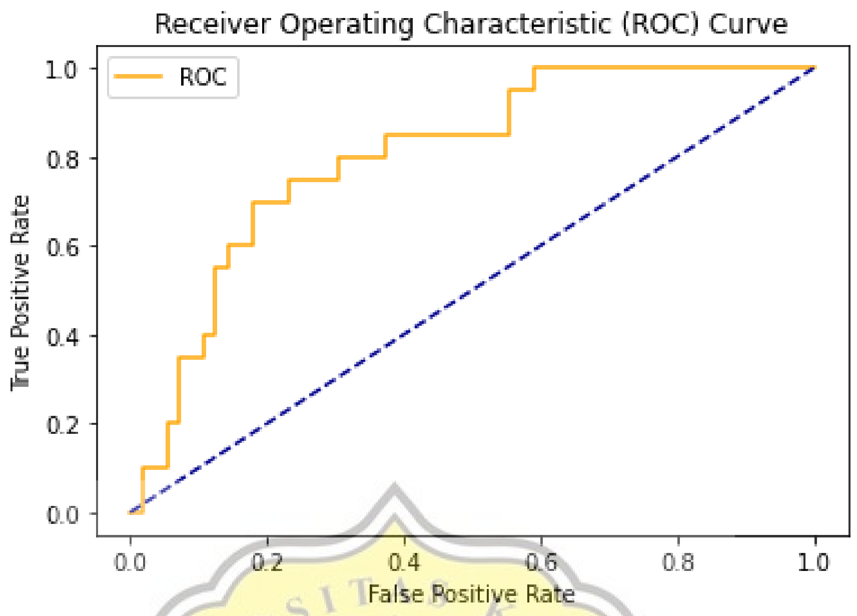


Illustration 5.21. *ROC Curve from the dataset*

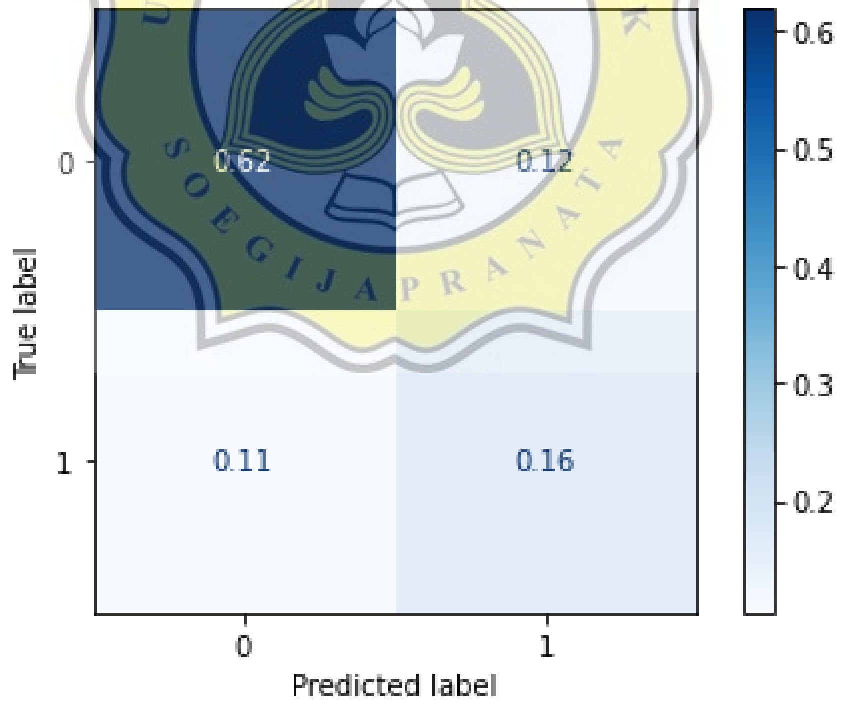
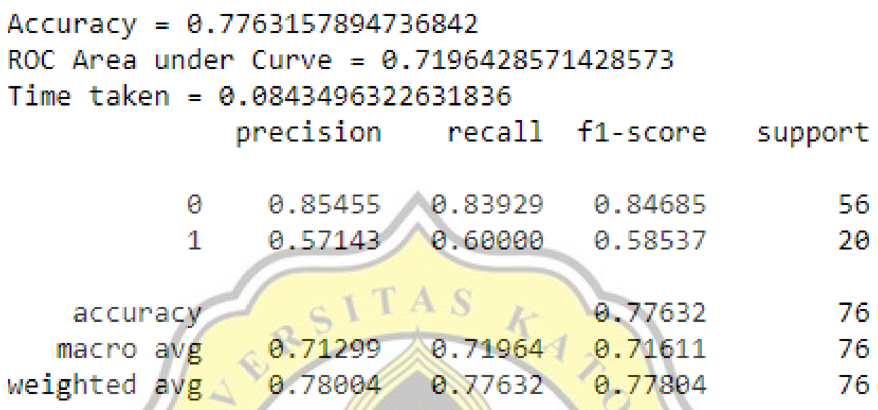


Illustration 5.22. *Confusion matrix from the dataset*

The ROC curve above is interpreting a good result of the test, it means that the model has works since the roc value of true positive rate is away from the diagonal. The confusion matrix also shows a good result for the model, from that when the Naïve Bayes using Sci-Kit learn implemented the accuracy is 0.77 as the result below.



```
Accuracy = 0.7763157894736842
ROC Area under Curve = 0.7196428571428573
Time taken = 0.0843496322631836
              precision    recall  f1-score   support

           0    0.85455   0.83929   0.84685        56
           1    0.57143   0.60000   0.58537        20

    accuracy                        0.77632        76
   macro avg    0.71299   0.71964   0.71611        76
weighted avg    0.78004   0.77632   0.77804        76
```

***Illustration** 5.23. The Accuracy Result of SciKit-Learn NB*

The Logistic regression without library is also tested. Before implement the algorithm, for the optimize and efficient model, the data evaluation must be done. First the MSE must be checked and it comes out as the images below.

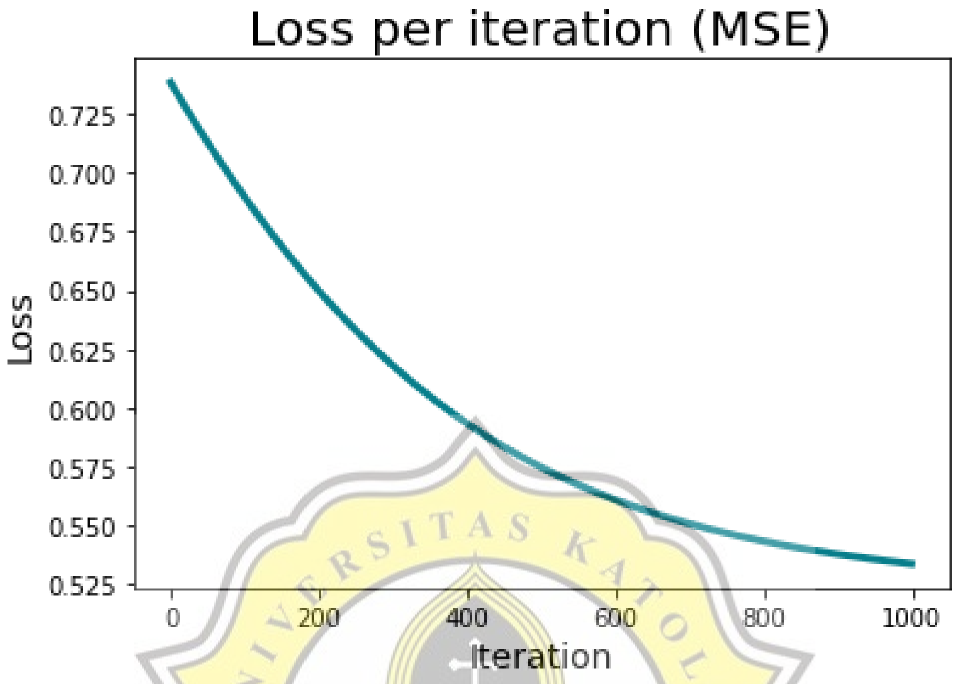


Illustration 5.24. *Mean Squared Errors of the model*

The MSE graph shows that the value is descending, it is a good point, because the more the iteration the lost become the minimal. After that, the most suitable learning rate must be found for the optimum logistic regression. Finding the best learning rate could be done by testing some value. It can be seen which one is the best learning rate by finding the minimal loss as the iteration bigger in the MSE. The graph is show below.

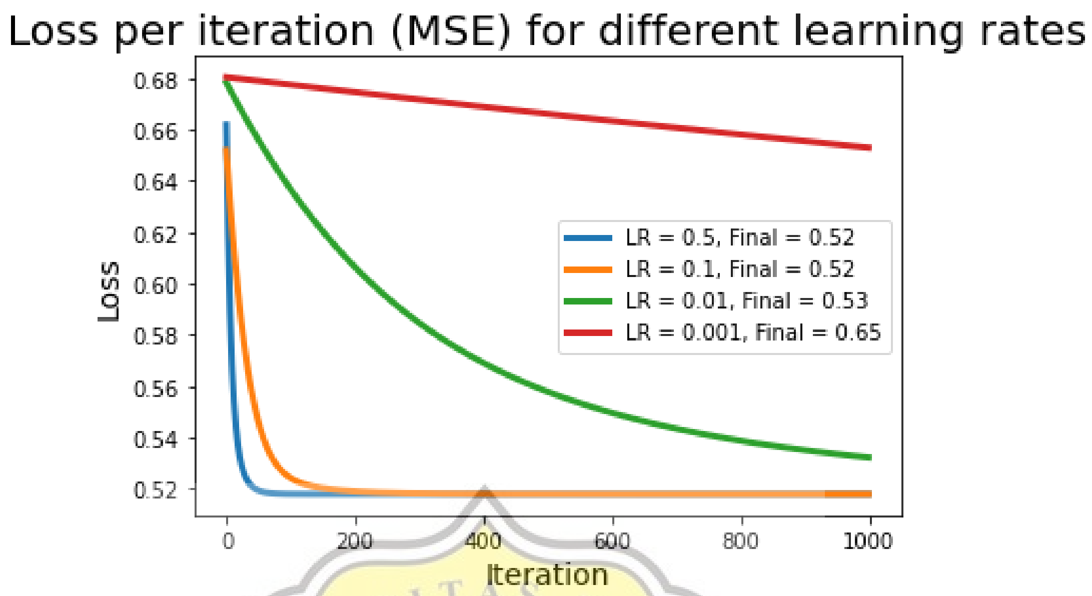
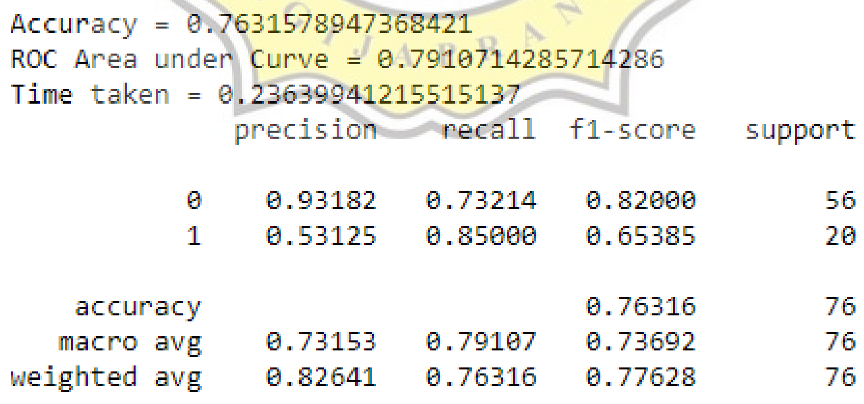


Illustration 5.25. *MSE for the best learning rate*

From the image above, the best learning rate is 0.5 since the line is descending and has the lowest loss value per iteration, so the logistic regression can be implemented with the learning rate 0.5.



```
Accuracy = 0.7631578947368421
ROC Area under Curve = 0.7910714285714286
Time taken = 0.23639941215515137
              precision    recall  f1-score   support

           0    0.93182   0.73214   0.82000        56
           1    0.53125   0.85000   0.65385        20

    accuracy                        0.76316        76
   macro avg    0.73153   0.79107   0.73692        76
weighted avg    0.82641   0.76316   0.77628        76
```

Illustration 5.26. *The Accuracy Result of LR Without Library*

The accuracy of this algorithm is 0.763. To show the accuracy, the same code is used to check the accuracy of the Linear Regression using Scikit algorithm. This code is the translation of the formula accuracy that has been discussed before.