
Presented by:

Anushka Prakash - 16ucs223
Kajol Choudhary - 16ucs224
Mahima Kejriwal - 16ucs225
Nireka Singhal - 16ucc062

Introduction to Data Science - Project

Classification problem using Pima Indian Diabetes Dataset

18th November 2018

OVERVIEW

The goals of our project are to develop accurate classifiers for data analysis of a dataset composed of several medical predictor (independent) variables and one target (dependent) variable, Outcome which tells us whether a patient is suffering from Diabetes or not. Independent variables include the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

GOALS

1. To build a machine learning model to accurately predict whether the patients in the dataset have diabetes or not.

ABOUT DATASET

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. For this project, **it is collected from [kaggle.com](https://www.kaggle.com)**.^[1] The objective of the dataset is to diagnostically predict whether or not 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 database consists of **768 rows** and **9 columns**.

Dataset Feature description:

Attributes	Measurement Unit	Type of attribute
Pregnancies	Number of times pregnant	Numeric
Glucose	Plasma glucose concentration a 2 hours in an oral glucose tolerance test	Numeric

Blood Pressure	Diastolic blood pressure (mm Hg)	Numeric
Skin Thickness	Triceps skin fold thickness (mm)	Numeric
Insulin	2-Hour serum insulin (mu U/ml)	Numeric
BMI	Body mass index (weight in kg/(height in m)^2)	Numeric
Diabetes Pedigree Function	Diabetes pedigree function	Numeric
Age	Age (years)	Numeric
Outcome (binary)	<i>Class variable</i> (0 or 1)	Numeric

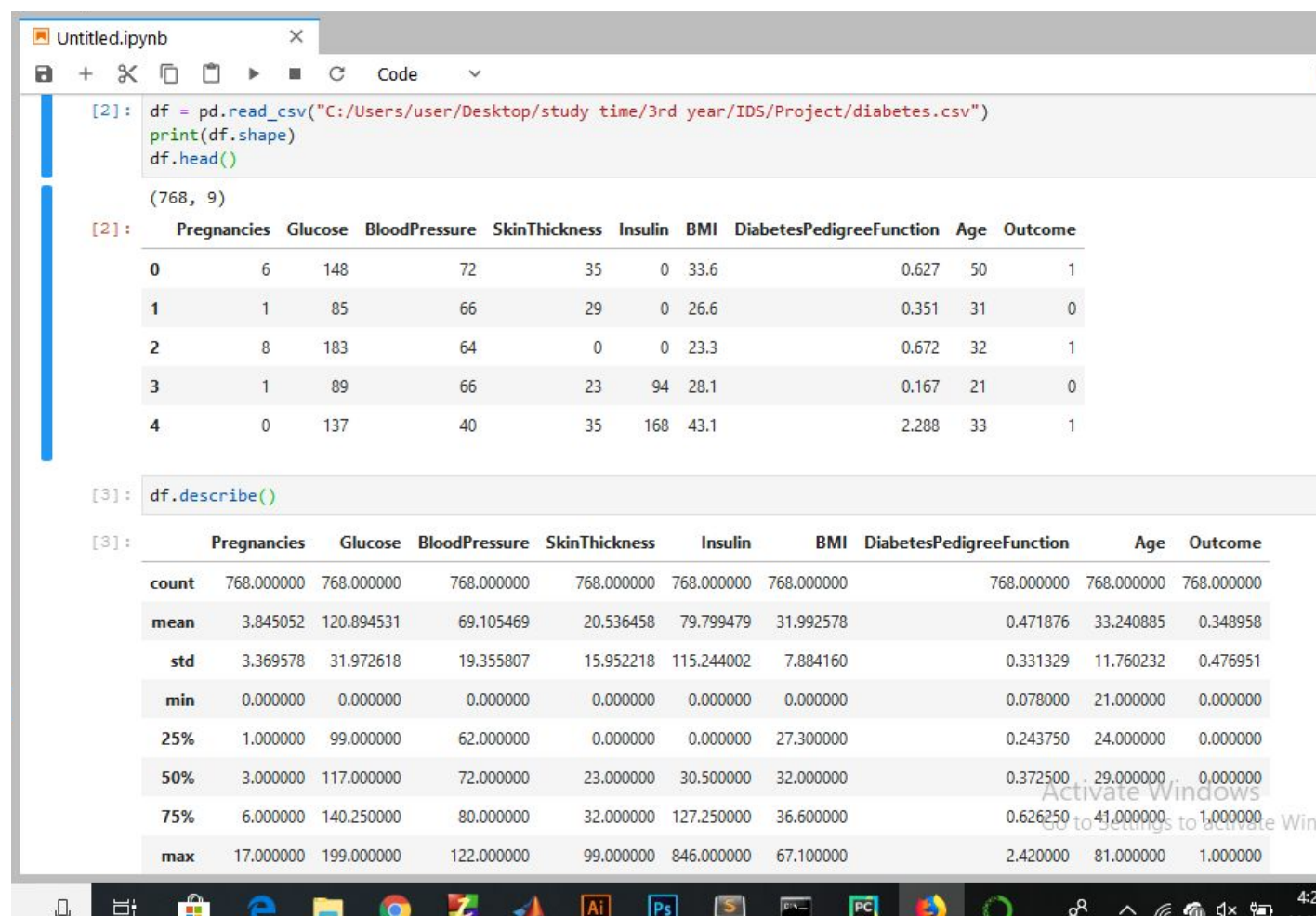


Figure 1 : Top 5 rows of the dataset and Basic Statistics of the Dataset.

DATA QUALITY PROBLEM

From Figure 1 we observed wrong data in some of the attributes which we later figured out to be missing values. The attributes which have missing values in the dataset are: Glucose , Blood Pressure , Skin Thickness , Insulin and BMI with value 0 written in their cells as shown in Figure 1.

We counted the number of rows for the above mentioned attributes with missing values and replaced them with NaN to make them identifiable. Refer Figure 2.

In the results, the number of rows with missing value for Insulin attribute is 374 and number of rows for Skin thickness attribute is 227 which are large given the total number of rows is 768, hence we cannot simply ignore these rows or eliminate them .Therefore, we estimated the mean of all the values in respective attributes to fill the missing values for the cells in those attributes.

The pregnancy attribute also has 0 value in some of its cells (which is valid) so we only replaced the above mentioned attributes missing values (0) by NaN.



```
IDS.ipynb
Python 3

[4]: #find no. of rows with missing values of relevent attribute
(df[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']]==0).sum()

[4]: Glucose      5
      BloodPressure  35
      SkinThickness 227
      Insulin      374
      BMI          11
      dtype: int64

[5]: #fill the missing values with NaNs to make them identifiable
df[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = df[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']]
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35.0	NaN	33.6	0.627	50	1
1	1	85.0	66.0	29.0	NaN	26.6	0.351	31	0
2	8	183.0	64.0	NaN	NaN	23.3	0.672	32	1
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1

Figure 2: It shows the count of missing values wrt. Attributes and then filled with NaNs to make them identifiable.

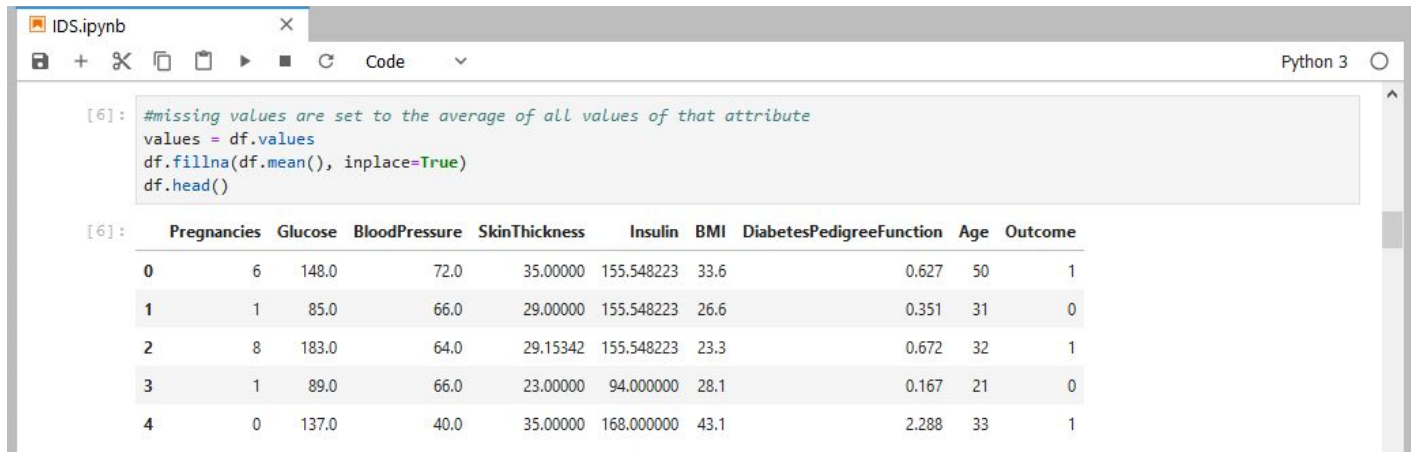


Figure 3: Nans are replaced with the mean of all the values of that attribute.

DATA VISUALIZATION

Visualizing the dataset in form of pie chart in figure 4.

Red denotes the percentage of people having diabetes and blue denotes the percentage of people without diabetes.

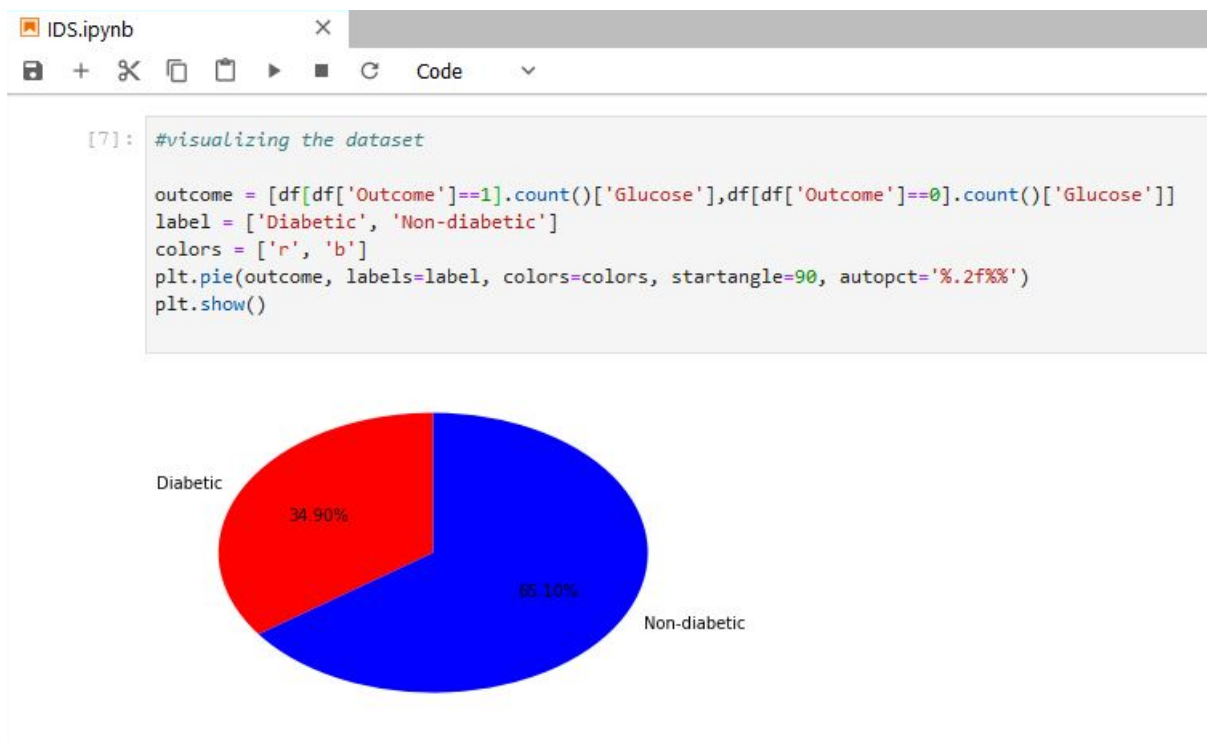
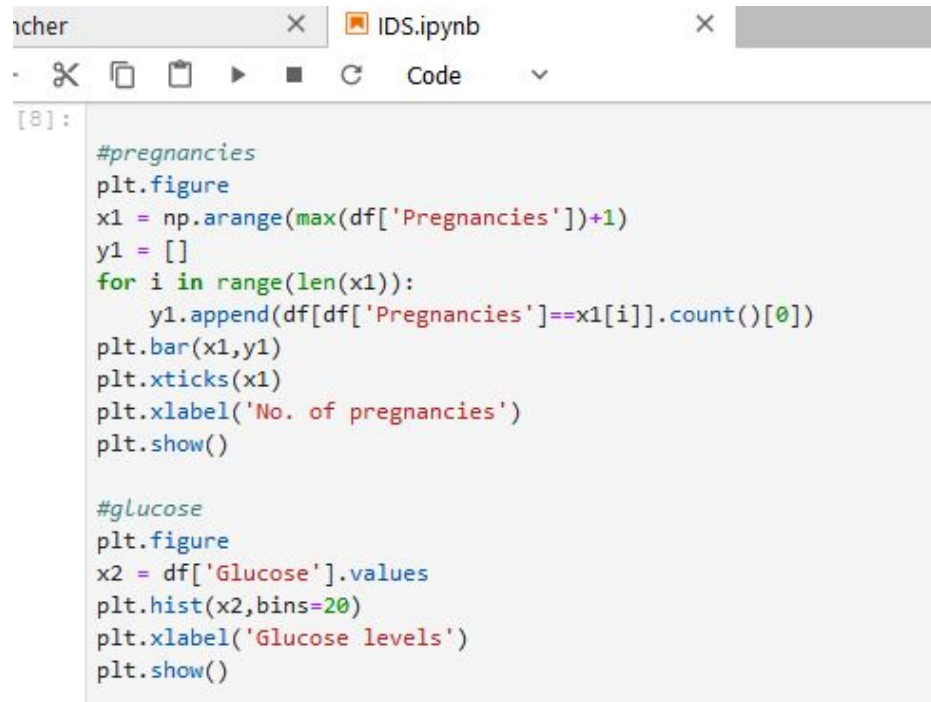


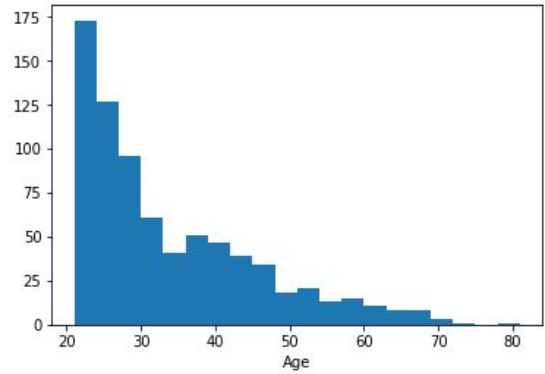
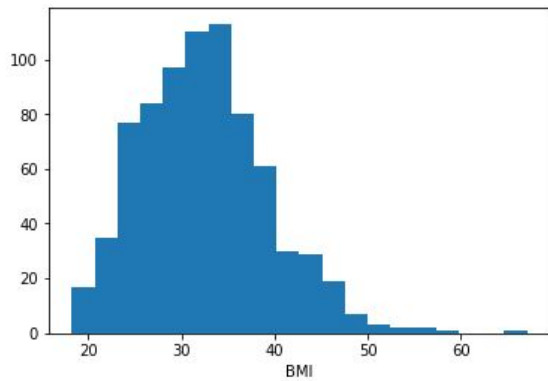
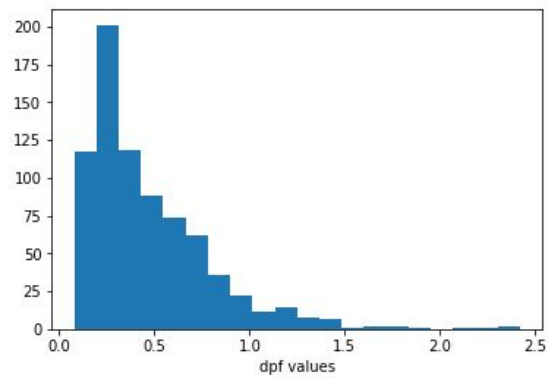
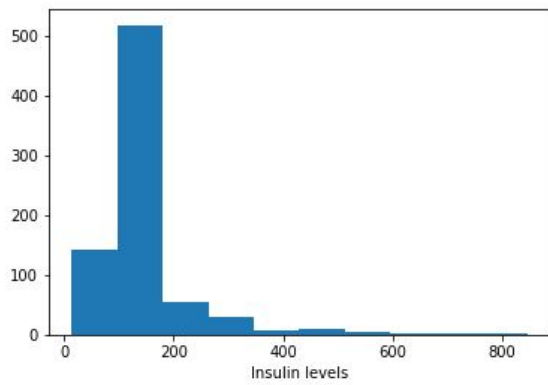
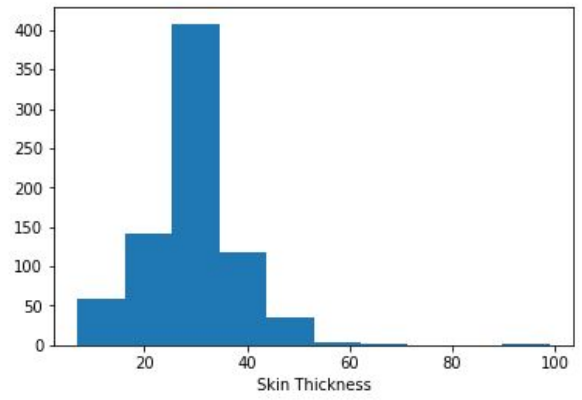
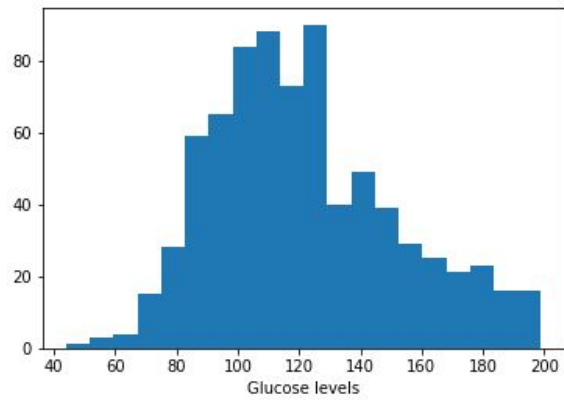
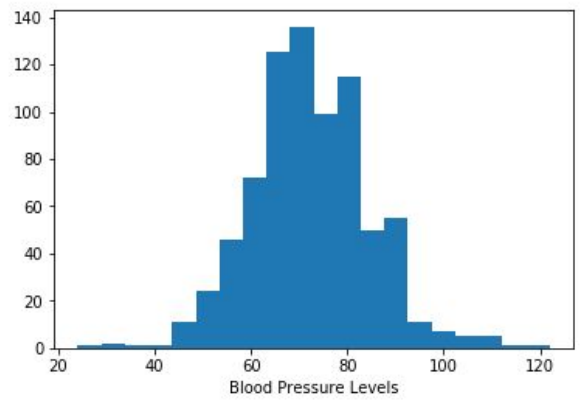
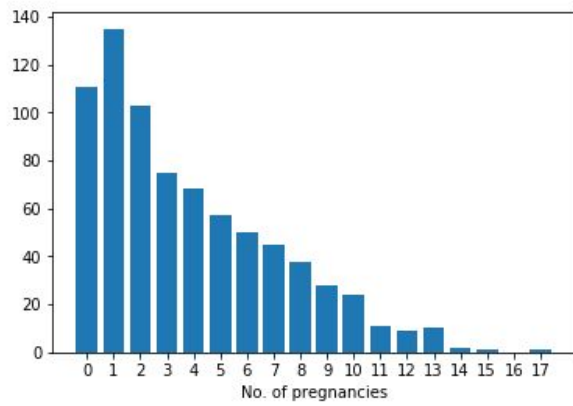
Figure 4: Visualizing the dataset and

We'll plot bar chart to show number of pregnancies since it is discrete quantity. And for the rest of the attributes namely Glucose , Blood Pressure , Skin Thickness , Insulin and BMI we'll plot histogram since they have continuous range values. Refer Figure 5 for the code and the series of figures after that for the chart of the respective attributes.



```
[8]:  
#pregnancies  
plt.figure  
x1 = np.arange(max(df['Pregnancies'])+1)  
y1 = []  
for i in range(len(x1)):  
    y1.append(df[df['Pregnancies']==x1[i]].count()[0])  
plt.bar(x1,y1)  
plt.xticks(x1)  
plt.xlabel('No. of pregnancies')  
plt.show()  
  
#glucose  
plt.figure  
x2 = df['Glucose'].values  
plt.hist(x2,bins=20)  
plt.xlabel('Glucose levels')  
plt.show()
```

Figure 5: The code to plot bar chart of #pregnancy attribute and histogram for glucose attribute.



DATA TRANSFORMATION

As the data of different attributes are varying in different ranges as shown in below screenshot ,we made it relevant by normalising the data(Min-Max normalization) to plot efficiently the respective attributes on x axis.

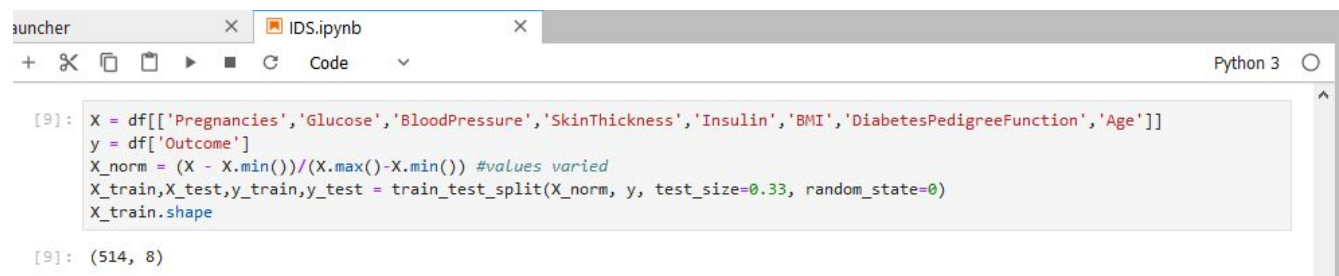
We use this formula to calculate the normalized X values.

Min-max normalization: to $[new_min_A, new_max_A]$

$$v' = \frac{v - min_A}{max_A - min_A} (new_max_A - new_min_A) + new_min_A$$

Splitting of the dataset into training and test data

We splitted the dataset into training and test data by giving one-third portion of data to test data and remaining to training data.



```
[9]: X = df[['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']]
y = df['Outcome']
X_norm = (X - X.min()) / (X.max() - X.min()) #values varied
X_train, X_test, y_train, y_test = train_test_split(X_norm, y, test_size=0.33, random_state=0)
X_train.shape

[9]: (514, 8)
```

Figure 3(II) : Normalizing and splitting the dataset into test data and training data.

In Figure 6 the classification of training data is shown . According to it, 35.8% of people are diabetic and 64.2% are non diabetic .

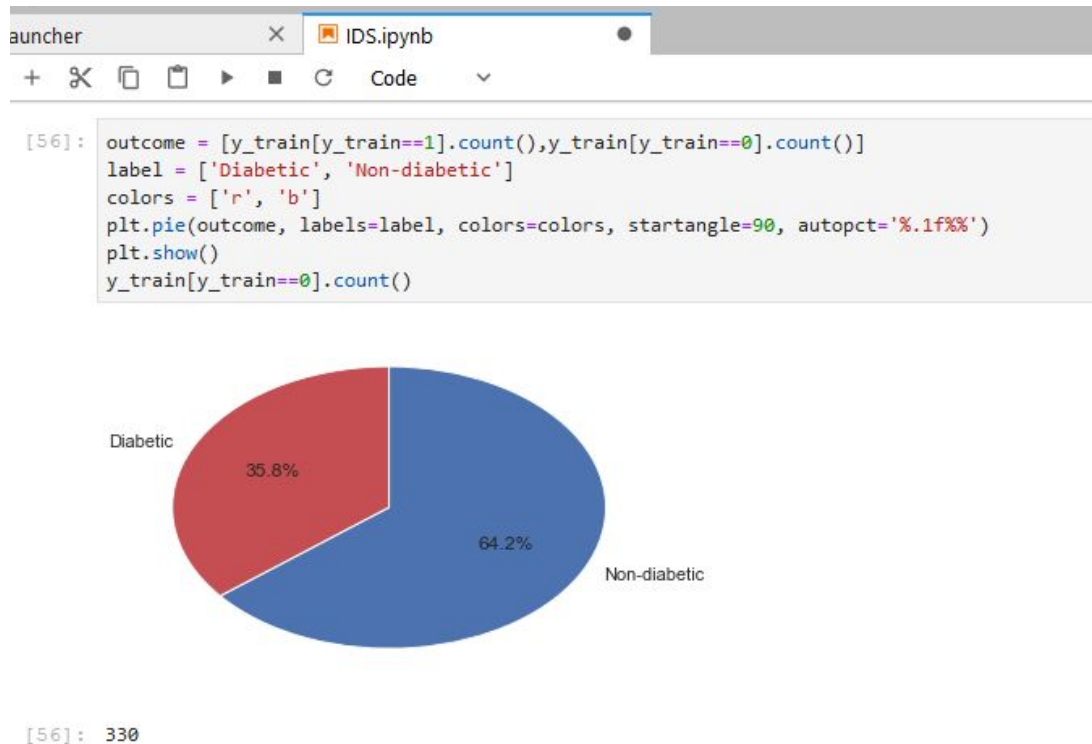


Figure 6 : Classification of the training dataset.

CORRELATION BETWEEN THE PAIR OF ATTRIBUTES

Figure 7 is the plots are between every pair of attributes. By analysing them we inferred that they are not correlated with each other ; hence we concluded that they are independent of each other.

PCA

Principal component analysis

We do the preprocessing by PCA .as we were having 8 dimensions which are difficult to visualize so we reduced it to two dimension to capture the largest amount of variation in data .

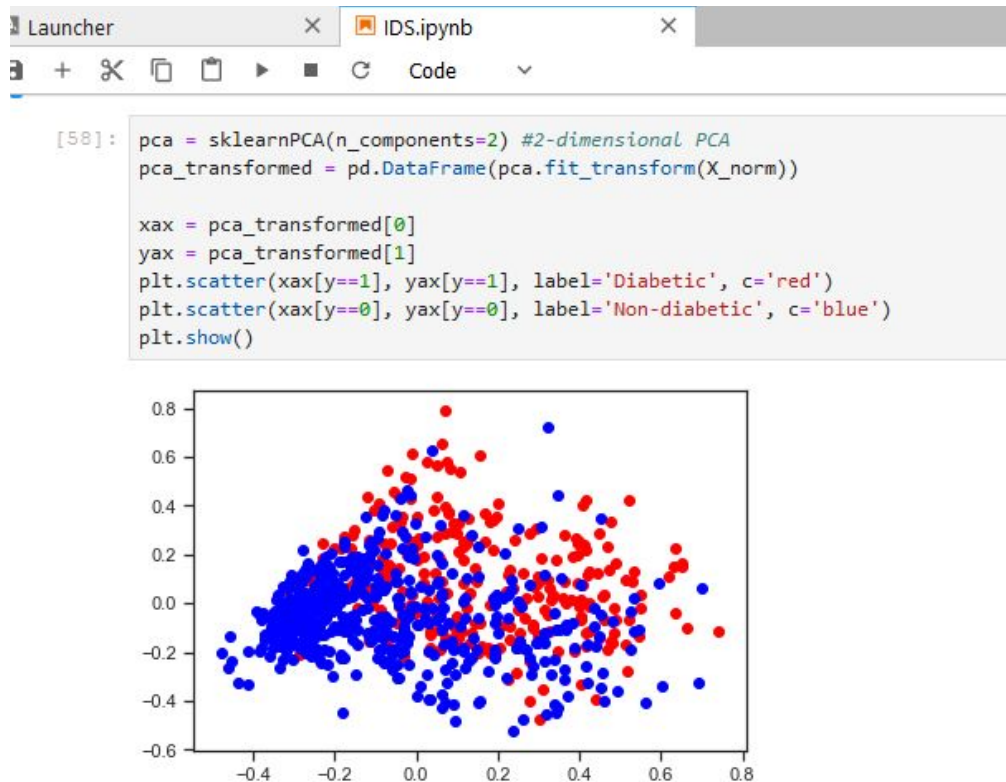


Figure 8: The PCA diagram to find variation in data. But no such clear variation is visible.

CLASSIFIERS AND EVALUATION MEASURES

Over the course of the last few weeks as we have been exploring the dataset, we realize that our initial assumptions had to be modified a bit in order to deal with the realities of the dataset itself. Several secondary questions emerged such with regard to effective dimensionality reduction via PCA. In the end, the process had multiple stages, including data pre-processing, sampling of training and test data and now the classification of that dataset.

PERFORMANCE/EVALUATION MEASURES:

Confusion matrix: A confusion matrix is a table used to describe the performance of a classification model on a set of test data for which the true values are known. It allows the visualization of the performance of an algorithm.

The Confusion matrix corresponding to our project is as follows :

	DIABETIC(Predicted)	NON DIABETIC(Predicted)
DIABETIC(Actual)	TRUE POSITIVE (TP)	FALSE NEGATIVE (FN)
NON DIABETIC(Actual)	FALSE POSITIVE (FP)	TRUE NEGATIVE (TN)

where,

TP : If patient is diabetic (actual) and is classified as Diabetic by the classifier.

TN : If patient is not diabetic (actual) and is classified as Non-Diabetic by the classifier.

FP : If patient is not diabetic (actual) and is classified as Diabetic by the classifier.

FN : If patient is diabetic (actual) and is classified as Non-Diabetic by the classifier.

Accuracy is one metric for evaluating classification models

. **Accuracy** is the fraction of predictions our model got right. $\text{Accuracy} = \text{Number of correct predictions} / \text{Total number of predictions}$.

For binary classification, accuracy can also be calculated in terms of positives and negatives as follows:

$$\text{Accuracy} = (TP + TN) / (TP + TN + FP + FN)$$

Our success metric for this project is to try and maximize the f-measure score:

The F1 score, commonly used in information retrieval, measures accuracy using the statistics *precision* p and *recall* r . Precision is the ratio of true positives (tp) to all predicted positives (tp + fp). Recall is the ratio of true positives to all actual positives (tp + fn). The F1 score is given by:

$$F1 = 2P \cdot R / (P + R) \text{ where } P = TP / (TP + FP), R = TP / (TP + FN)$$

The F1 metric weights recall and precision equally, and a good retrieval algorithm will maximize both precision and recall simultaneously. Thus, moderately good performance on both will be favored over extremely good performance on one and poor performance on the other.

The CLASSIFIERS we used in this project for classification are as follows:

1.Support Vector Machine :

SVM is a discriminative classifier formally defined by a separating hyperplane,given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples.

Since we have a 2 dimensional plane so it gives a line dividing the plane in two parts where each class lay in either side. Figure 9(a) is showing the data validation by SVM classifier

2.Naive Bayes :

It is a supervised machine-learning algorithm that uses the Bayes' Theorem, which assumes that features are statistically independent. The theorem relies on the *naïve* assumption that input variables are independent of each other, i.e. there is no way to know anything about other variables when given an additional variable. Regardless of this assumption, it has proven itself to be a classifier with good results. Refer Figure 9(b).

3.Random Forest

It is ensemble algorithm .It creates a set of decision trees from randomly selected subset of training set. It then aggregates the votes from different decision trees to decide the final class of the test object. Refer Figure 9(3).

Comparison Between the classifiers w.r.t. their Accuracy and F score:

	SVM	NAIVE BAYES	RANDOM FOREST
Accuracy	77.95%	75.19%	77.56%
Error rate	22.05%	24.71%	22.44%
F score	57.57%	58.27%	57.77%
Time Executed (in millisec)	0.08105 sec	0.04686 sec	0.08661 sec

Accuracy : SVM > Random Forest > Naive Bayes

F score : Naive Bayes > Random Forest > SVM

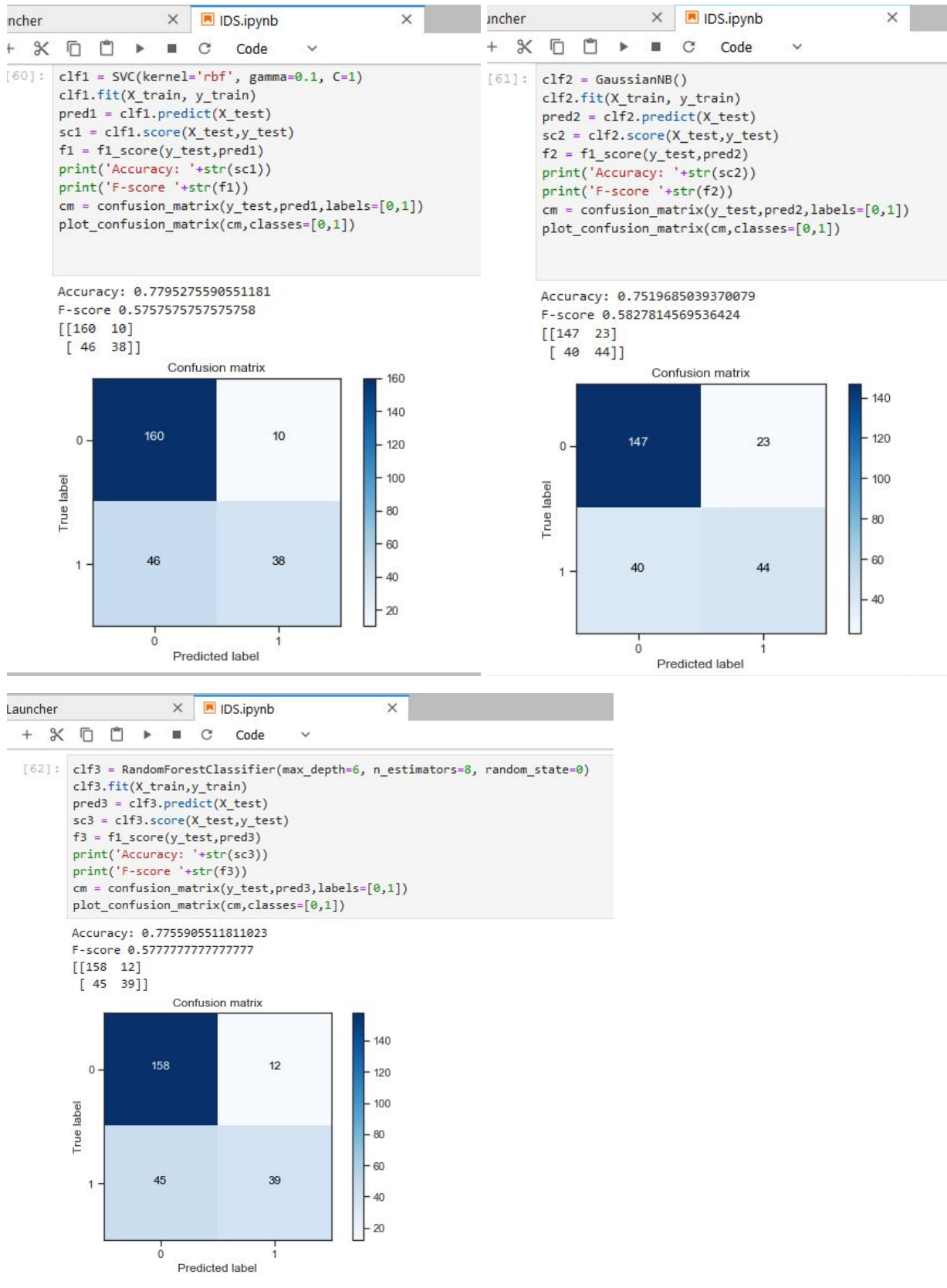


Figure 9: a) Support Vector Machine, b) Naive Bayes, c) Random Forest Classifiers.

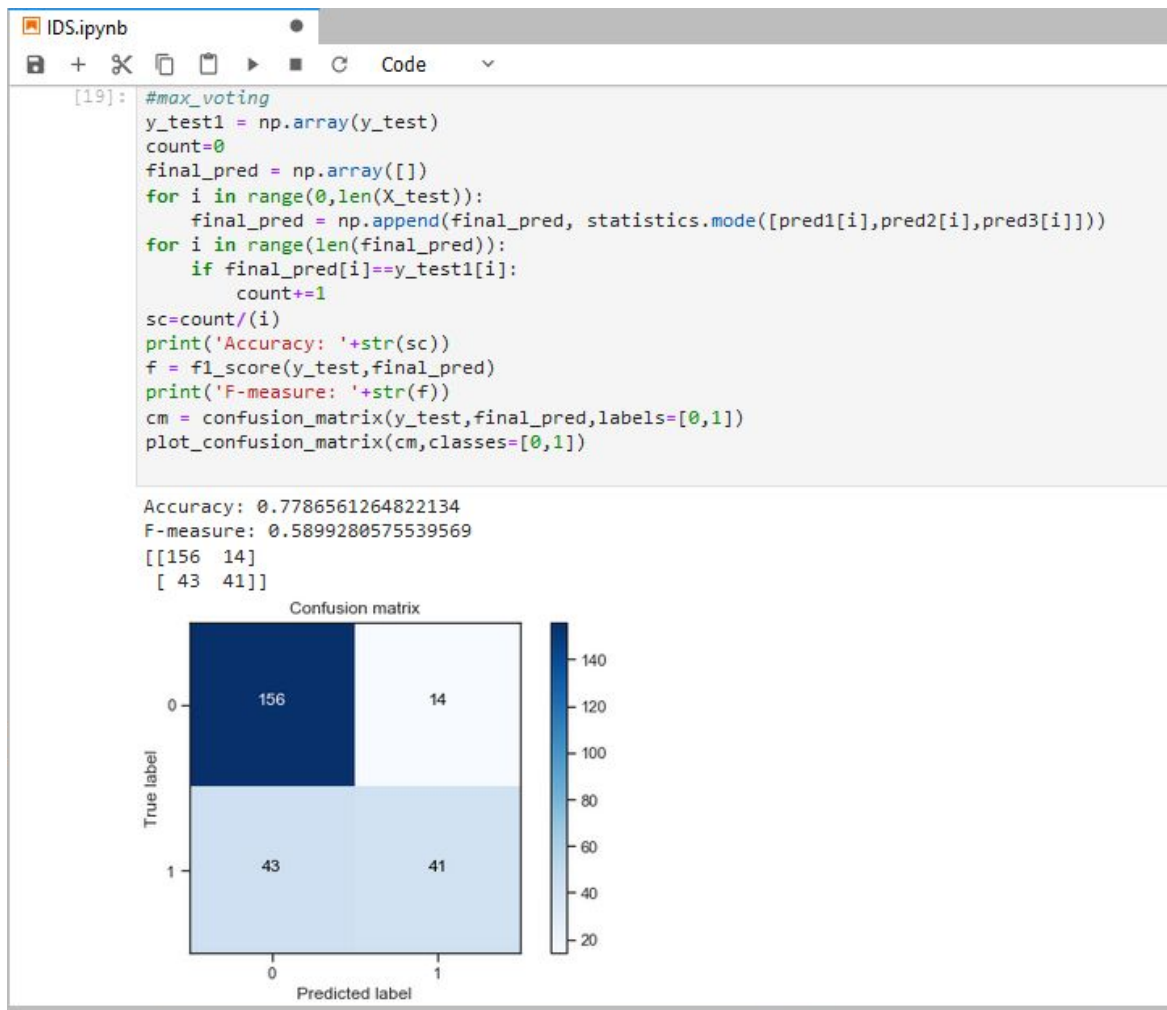
MAJORITY VOTING

After classifying the dataset we applied Maximum voting(Hard). Hard voting is the simplest case of majority voting. Here, we predict the class label via majority (plurality) voting of each classifier :

We combined the three classifiers that classify our training sample as follows:

- SVM (classifier 1) -> Diabetic (class 0)
- Naive Bayes (classifier 2) -> Diabetic (class 0)
- Random Forest (classifier 3) -> Non-Diabetic (class 1)

Via majority vote, we would classify the sample as "Diabetic".



FINAL ANALYSIS

In the final analysis, we learned a great deal about the dataset that we collected for this project. We had a few questions at the start of this project, of which the most important was: “Can we build a machine learning model to accurately predict whether the patients in the dataset have diabetes or not”

The answer to this question is Yes, provided that class attribute is balanced, if we face class imbalance problem then we need to do stratified cross validation (sampling) for training and test data at the time of splitting the dataset and then apply the classifiers.

This project, rather than advancing new theories about data analysis, was more about maximizing a particular success metric in a competitive context. Nevertheless, we gained significant insight into the nature of different classification schemes and are now well-equipped to continue tackling related challenges in the domain of classification into the future.

REFERENCES

[1] <http://stackexchange.com/about>

[2] <https://www.kaggle.com/uciml/pima-indians-diabetes-database/version/1>

[3] S. Raschka. [Python Machine Learning](#). Packt Publishing Ltd., 2015.

[4] <https://scikit-learn.org/stable/modules/sgd.html>

[5] Code for reference.(Next Page)

CODE for Reference :

```
import numpy as np

import pandas as pd

import statistics

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import GaussianNB

from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from sklearn.decomposition import PCA as sklearnPCA

from sklearn import tree

from sklearn.metrics import precision_score, recall_score, confusion_matrix, f1_score

from sklearn import metrics

import itertools


df = pd.read_csv("C:/Users/user/Desktop/study/3rd year/IDS/Project/diabetes.csv")

print(df.shape)

df.head()


df.describe()
```

```
#find no. of rows with missing values of relevent attribute
```

```
(df[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']]==0).sum()
```

```
#fill the missing values with NaNs to make them identifiable
```

```
df[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] =
```

```
df[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']].replace(0,np.NaN)
```

```
df.head()
```

```
#missing values are set to the average of all values of that attribute
```

```
values = df.values
```

```
df.fillna(df.mean(), inplace=True)
```

```
df.head()
```

```
#visualizing the dataset
```

```
outcome = [df[df['Outcome']==1].count()['Glucose'],df[df['Outcome']==0].count()['Glucose']]
```

```
label = ['Diabetic', 'Non-diabetic']
```

```
colors = ['r', 'b']
```

```
plt.pie(outcome, labels=label, colors=colors, startangle=90, autopct='%2f%%')
```

```
plt.show()
```

```
#pregnancies
```

```
plt.figure
```

```
x1 = np.arange(max(df['Pregnancies'])+1)
```

```
y1 = []
```

```
for i in range(len(x1)):
```

```
    y1.append(df[df['Pregnancies']==x1[i]].count()[0])
```

```
plt.bar(x1,y1)
```

```
plt.xticks(x1)
```

```
plt.xlabel('No. of pregnancies')
```

```
plt.show()
```

```
#glucose
```

```
plt.figure
```

```
x2 = df['Glucose'].values
```

```
plt.hist(x2,bins=20)
```

```
plt.xlabel('Glucose levels')
```

```
plt.show()
```

```
#blood pressure
```

```
plt.figure
```

```
x3 = df['BloodPressure'].values
```

```
plt.hist(x3,bins=20)
```

```
plt.xlabel('Blood Pressure Levels')
```

```
plt.show()
```

```
#Skin Thickness
```

```
plt.figure
```

```
x4 = df['SkinThickness'].values
```

```
plt.hist(x4,bins=10)
```

```
plt.xlabel('Skin Thickness')
```

```
plt.show()
```

```
#insulin
```

```
plt.figure
```

```
x5 = df['Insulin'].values
```

```
plt.hist(x5,bins=10)
```

```
plt.xlabel('Insulin levels')
```

```
plt.show()
```

```
#bmi
```

```
plt.figure
```

```
x6 = df['BMI'].values
```

```
plt.hist(x6,bins=20)
```

```
plt.xlabel('BMI')
```

```
plt.show()
```

```
#dpf
```

```
plt.figure
```

```
x7 = df['DiabetesPedigreeFunction'].values
```

```
plt.hist(x7,bins=20)
```

```
plt.xlabel('dpf values')
```

```
plt.show()
```

```
#age
```

```
plt.figure
```

```
x8 = df['Age'].values
```

```
plt.hist(x8,bins=20)
```

```
plt.xlabel('Age')
```

```
plt.show()
```

```
X = df[['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age']]
```

```
y = df['Outcome']
```

```
X_norm = (X - X.min())/(X.max()-X.min()) #values varied
```

```
X_train,X_test,y_train,y_test = train_test_split(X_norm, y, test_size=0.33, random_state=0)
```

```
X_train.shape
```

```
outcome = [y_train[y_train==1].count(),y_train[y_train==0].count()]
```

```
label = ['Diabetic', 'Non-diabetic']
```

```
colors = ['r', 'b']
```

```
plt.pie(outcome, labels=label, colors=colors, startangle=90, autopct='%.1f%%')
```

```
plt.show()
```

```
y_train[y_train==0].count()
```

```
plt.figure
```

```
sns.set(style='ticks')
```

```
sns.pairplot(df, hue='Outcome')
```

```
plt.show()
```

```
pca = sklearnPCA(n_components=2) #2-dimensional PCA
```

```
pca_transformed = pd.DataFrame(pca.fit_transform(X_norm))
```

```
xax = pca_transformed[0]
```

```
yax = pca_transformed[1]
```

```
plt.scatter(xax[y==1], yax[y==1], label='Diabetic', c='red')
```

```
plt.scatter(xax[y==0], yax[y==0], label='Non-diabetic', c='blue')
```

```
plt.show()
```

```
def plot_confusion_matrix(cm, classes, title='Confusion matrix', cmap=plt.cm.Blues):
```

```
    plt.figure()
```

```
    print(cm)
```

```
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
```

```
    plt.title(title)
```

```
    plt.colorbar()
```

```
    tick_marks = np.arange(len(classes))
```

```
    plt.xticks(tick_marks, classes)
```

```
    plt.yticks(tick_marks, classes)
```

```
    fmt = 'd'
```

```
    thresh = cm.max() / 2.
```

```
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
```

```
    plt.text(j, i, format(cm[i, j], fmt),
```

```
            horizontalalignment="center",
```

```
            color="white" if cm[i, j] > thresh else "black")
```

```
plt.tight_layout()
```

```
plt.ylabel('True label')
```

```
plt.xlabel('Predicted label')
```

```
clf1 = SVC(kernel='rbf', gamma=0.1, C=1)
```

```
clf1.fit(X_train, y_train)
```

```
pred1 = clf1.predict(X_test)
```

```
sc1 = clf1.score(X_test, y_test)
```

```
f1 = f1_score(y_test, pred1)
```

```
print('Accuracy: ' + str(sc1))
```

```
print('F-score ' + str(f1))
```

```
cm = confusion_matrix(y_test, pred1, labels=[0, 1])
```

```
plot_confusion_matrix(cm, classes=[0, 1])
```

```
clf2 = GaussianNB()
```

```
clf2.fit(X_train, y_train)
```

```
pred2 = clf2.predict(X_test)
```

```
sc2 = clf2.score(X_test, y_test)
```

```
f2 = f1_score(y_test, pred2)
```

```
print('Accuracy: '+str(sc2))

print('F-score '+str(f2))

cm = confusion_matrix(y_test,pred2,labels=[0,1])

plot_confusion_matrix(cm,classes=[0,1])


clf3 = RandomForestClassifier(max_depth=6, n_estimators=8, random_state=0)

clf3.fit(X_train,y_train)

pred3 = clf3.predict(X_test)

sc3 = clf3.score(X_test,y_test)

f3 = f1_score(y_test,pred3)

print('Accuracy: '+str(sc3))

print('F-score '+str(f3))

cm = confusion_matrix(y_test,pred3,labels=[0,1])

plot_confusion_matrix(cm,classes=[0,1])

#max_voting

y_test1 = np.array(y_test)

count=0

final_pred = np.array([])

for i in range(0,len(X_test)):

    final_pred = np.append(final_pred, statistics.mode([pred1[i],pred2[i],pred3[i]]))

for i in range(len(final_pred)):

    if final_pred[i]==y_test1[i]:

        count+=1

sc=count/(i)
```

```
print('Accuracy: '+str(sc))
```

```
f = f1_score(y_test,final_pred)
```

```
print('F-measure: '+str(f))
```

```
cm = confusion_matrix(y_test,final_pred,labels=[0,1])
```

```
plot_confusion_matrix(cm,classes=[0,1])
```