Diabetes Classification Report

A. Title

Pima Indians Diabetes Database. This dataset is from the National Institute of Diabetes and Digestive and Kidney Diseases and released by Johns Hopkins University.

B. Data description

There are number of 768 instances and 8 attributes with Class. All the attributes are numeric and the Class is categorical. The attributes are listed below:

1. Number of times pregnant
2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. Diastolic blood pressure (mm Hg)
4. Triceps skin fold thickness (mm)
5. 2-Hour serum insulin (mu U/ml)
6. Body mass index (weight in kg/(height in m)^2)
7. Diabetes pedigree function
8. Age (years)
9. Class variable (0 or 1)

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.

C. Task

Classify a person tests positive or negative for diabetes based on the 8 attributes.

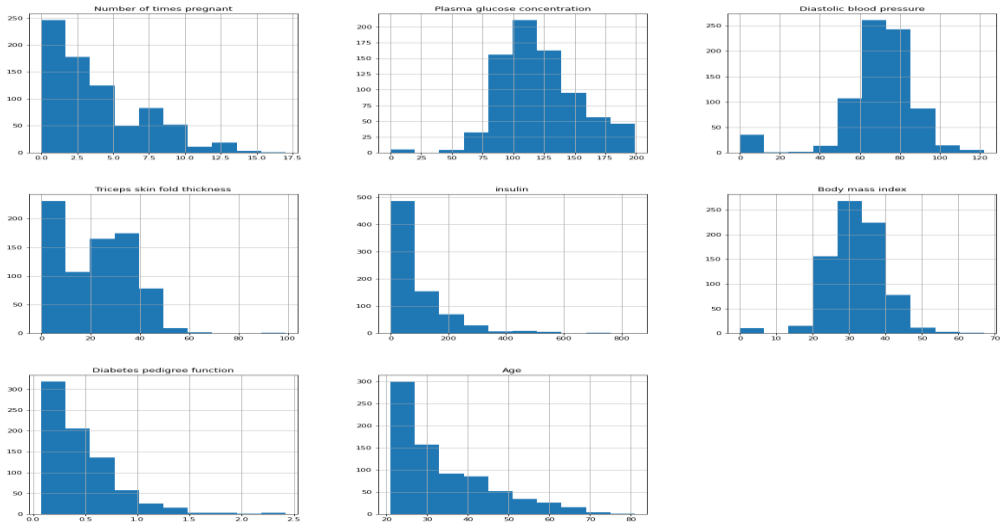
D. Data Information

The figure below shows the first five information of the diabetes dataset (diabetes.head()).

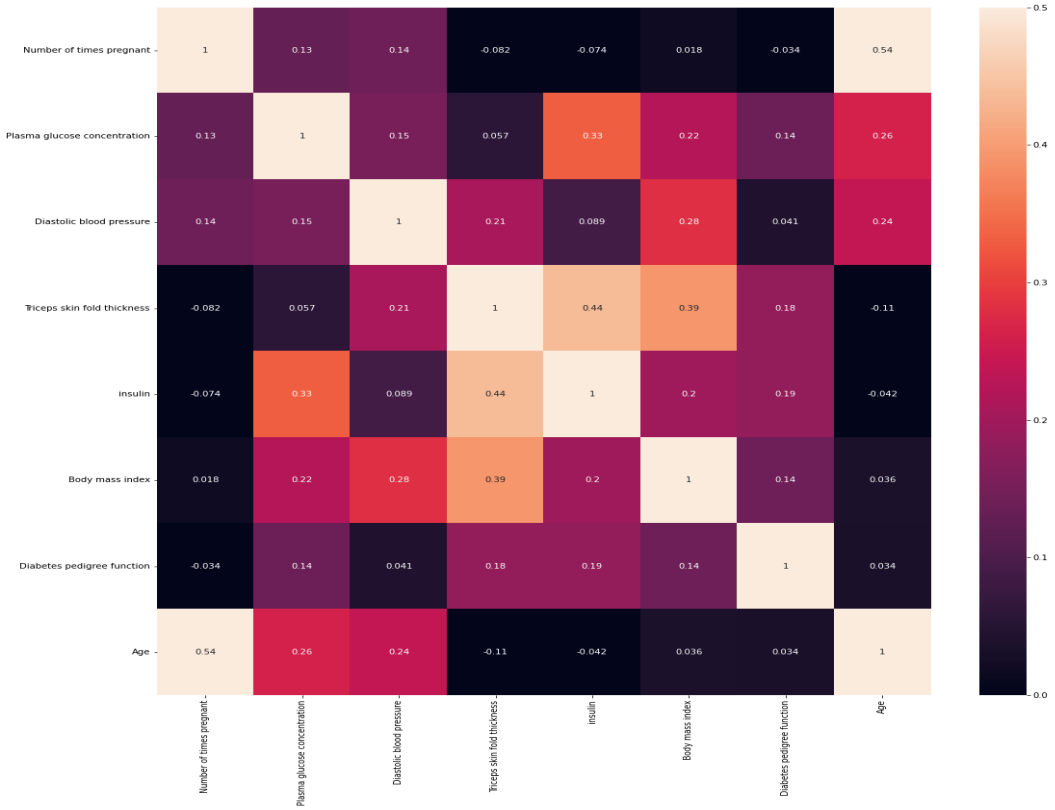
	Number of times pregnant	Plasma glucose concentration	Diastolic blood pressure	Triceps skin fold thickness	insulin	Body mass index	Diabetes pedigree function	Age	Class
0	6	148	72	35	0	33.6	0.627	50	tested_positive
1	1	85	66	29	0	26.6	0.351	31	tested_negative
2	8	183	64	0	0	23.3	0.672	32	tested_positive
3	1	89	66	23	94	28.1	0.167	21	tested_negative
4	0	137	40	35	168	43.1	2.288	33	tested_positive

E. Data Visualization

The best way to start the analysis is visualizing the dataset. Below is the histogram plot showing the distribution count of each attribute.



Also, the correlation between each attributes can be visualized using heatmap. From the output, the lighter colors indicate more correlation. There is correlation between pairs of features, like age and pregnancies, or BMI and skin thickness, etc.



F. Data Cleaning

First, I checked for missing values using the isnull().sum() function.

```
diabetes.isnull().sum()
```

```
Number of times pregnant          0
Plasma glucose concentration      0
Diastolic blood pressure          0
Triceps skin fold thickness       0
insulin                           0
Body mass index                   0
Diabetes pedigree function        0
Age                               0
Class                             0
dtype: int64
```

Initially, it seems there is no missing value but Glucose, Blood Pressure, Insulin, Triceps Skin Thickness and BMI contain a value of 0 which may represent the missing value. For example, a person's glucose or insulin value cannot be 0. Considering this situation, let's assign the 0 values

to the relevant values as NaN and then apply the operations to the missing values. However, the pregnancy value can be 0 so it is excluded from the NaN.

	Number of times pregnant	Plasma glucose concentration	Diastolic blood pressure	Triceps skin fold thickness	insulin	Body mass index	Diabetes pedigree function	Age	Class
0	6	148.0	72.0	35.0	NaN	33.6	0.627	50	tested_positive
1	1	85.0	66.0	29.0	NaN	26.6	0.351	31	tested_negative
2	8	183.0	64.0	NaN	NaN	23.3	0.672	32	tested_positive
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	tested_negative
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	tested_positive

Then, the NaN values are replaced with the mean value using the simple imputer from sklearn.

	Number of times pregnant	Plasma glucose concentration	Diastolic blood pressure	Triceps skin fold thickness	insulin	Body mass index	Diabetes pedigree function	Age	Class
0	6	148.0	72.0	35.00000	155.548223	33.6	0.627	50.0	tested_positive
1	1	85.0	66.0	29.00000	155.548223	26.6	0.351	31.0	tested_negative
2	8	183.0	64.0	29.15342	155.548223	23.3	0.672	32.0	tested_positive
3	1	89.0	66.0	23.00000	94.000000	28.1	0.167	21.0	tested_negative
4	0	137.0	40.0	35.00000	168.000000	43.1	2.288	33.0	tested_positive

For the Class categorical attribute, one hot encoding is used to change the outcome from “tested_negative” and “tested_positive” to 0 or 1.

	Number of times pregnant	Plasma glucose concentration	Diastolic blood pressure	Triceps skin fold thickness	insulin	Body mass index	Diabetes pedigree function	Age	tested_negative	tested_positive
0	6	148.0	72.0	35.00000	155.548223	33.6	0.627	50.0	0.0	1.0
1	1	85.0	66.0	29.00000	155.548223	26.6	0.351	31.0	1.0	0.0
2	8	183.0	64.0	29.15342	155.548223	23.3	0.672	32.0	0.0	1.0
3	1	89.0	66.0	23.00000	94.000000	28.1	0.167	21.0	1.0	0.0
4	0	137.0	40.0	35.00000	168.000000	43.1	2.288	33.0	0.0	1.0

G. Data Preprocessing

Before implementing classification algorithm, the dataset is scaled using sklearn's MinMaxScaler() function. This function transform features by scaling each feature to a given range.

```
min_max_scaler = preprocessing.MinMaxScaler()
scaled_data = min_max_scaler.fit_transform(diabetesnp)
scaled_df=pd.DataFrame(scaled_data)
scaled_df
```

	0	1	2	3	4	5	6	7	8	9
0	0.352941	0.743719	0.590164	0.353535	0.000000	0.500745	0.234415	0.483333	0.0	1.0
1	0.058824	0.427136	0.540984	0.292929	0.000000	0.396423	0.116567	0.166667	1.0	0.0
2	0.470588	0.919598	0.524590	0.000000	0.000000	0.347243	0.253629	0.183333	0.0	1.0
3	0.058824	0.447236	0.540984	0.232323	0.111111	0.418778	0.038002	0.000000	1.0	0.0
4	0.000000	0.688442	0.327869	0.353535	0.198582	0.642325	0.943638	0.200000	0.0	1.0

H. Train Test Split

Using sklearn's train_test_split, the feature (X) and target (y) dataframes is split into a training set (80%) and testing set (20%). Training set is used for building classification models and testing set is used for evaluating the performance of the model.

```
X = df_scaled.loc[:,['Number of times pregnant','Plasma glucose concentration', 'Diastolic blood pressure', 'Triceps skin fold thickness', 'insulin',
y = df_scaled.loc[:,['tested_positive']]
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

print(X_train.shape, X_test.shape)
print(y_train.shape, y_test.shape)
```

```
(614, 8) (154, 8)
(614, 1) (154, 1)
```

I. Justification

1. Decision Tree Classifier

- It can handle various data type i.e. discrete or continuous values.
- Requires little or no data preparation.

2. Logistic Regression

- Used for binary response i.e. it has only two possible outcomes (e.g. 0 or 1).
- Easy to implement, interpret, and very efficient to train.

J. Decision Tree Classifier Performance Evaluation and Result

From sklearn metrics, the accuracy score, confusion matrix and classification report were imported for the model performance evaluation.

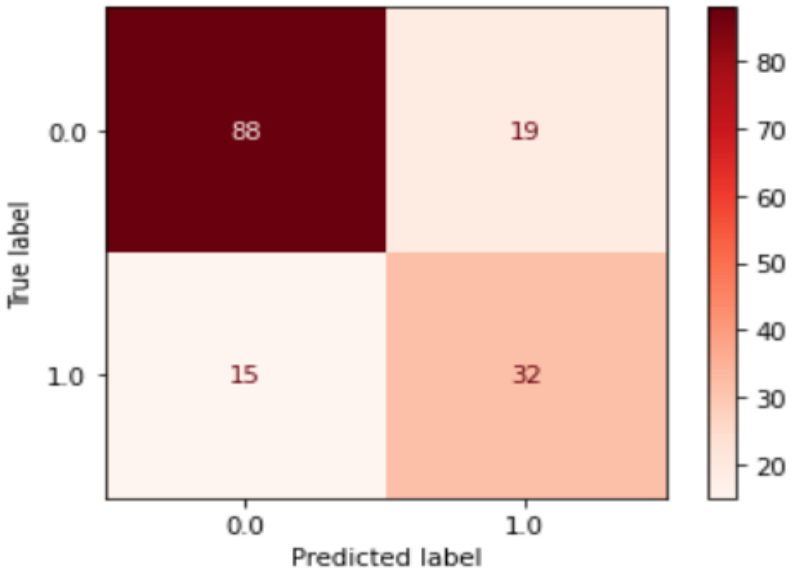
1. Accuracy Score

```
acc_tree = accuracy_score(y_test, y_pred)

print(f'The accuracy for decision tree is {round(100 * acc_tree, 2)}%')
```

```
The accuracy for decision tree is 77.92%
```

2. Confusion Matrix



True Positives (TP [1, 1]): the model correct prediction of people that have diabetes = 32

True Negatives (TN [0, 0]): the model correct prediction of people that don't have diabetes = 88

False Positives (FP [0, 1]): the model incorrect prediction of people that do have diabetes (a "Type I error") = 19

False Negatives (FN [1, 0]): the model incorrect prediction of people that do not have diabetes (a "Type II error") = 15

3. Classification Report

```
report = classification_report(y_test,  y_pred)
print(report)
```

```
              precision    recall  f1-score   support

         0.0       0.85      0.82      0.84       107
         1.0       0.63      0.68      0.65        47

    accuracy                           0.78       154
   macro avg       0.74      0.75      0.75       154
weighted avg       0.79      0.78      0.78       154
```

- Precision is the accuracy of positive predictions. Precision = TP/ (TP + FP). The model has a high precision of 0.85 and 0.63.
- Recall is the fraction of positives that were correctly identified. Recall = TP/ (TP+FN). The model has a high recall of 0.82 and 0.68.
- The F1 score is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0. The model F1 score is 0.84 and 0.65.
- Support is the number of actual occurrences of the class in the specified dataset.

K. <u>Logistic Regression Performance Evaluation and Result</u>

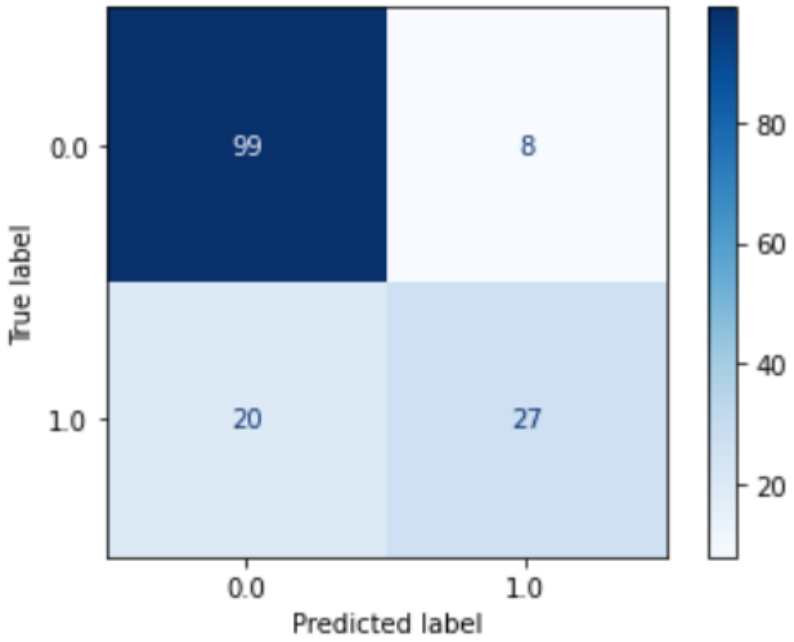
1. Accuracy Score

```
acc_lr = accuracy_score(y_test, y_pred_lr)

print(f'The accuracy for Logistic Regression is {round(100 * acc_lr, 2)}%' )
```

```
The accuracy for Logistic Regression is 81.82%
```

2. Confusion Matrix



True Positives (TP [1, 1]): the model correct prediction of people that have diabetes = 27

True Negatives (TN [0, 0]): the model correct prediction of people that don't have diabetes = 99

False Positives (FP [0, 1]): the model incorrect prediction of people that do have diabetes (a "Type I error") = 8

False Negatives (FN [1, 0]): the model incorrect prediction of people that do not have diabetes (a "Type II error") = 20

3. Classification Report

```
report_lr = classification_report(y_test,  y_pred_lr)
print(report_lr)
```

```
              precision    recall  f1-score   support

         0.0       0.83      0.93      0.88       107
         1.0       0.77      0.57      0.66        47

    accuracy                           0.82       154
   macro avg       0.80      0.75      0.77       154
weighted avg       0.81      0.82      0.81       154
```

- Precision is the accuracy of positive predictions. Precision = TP/ (TP + FP). The model has a high precision of 0.83 and 0.77.
- Recall is the fraction of positives that were correctly identified. Recall = TP/ (TP+FN). The model has a high recall of 0.93 and 0.57.
- The F1 score is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0. The model F1 score is 0.88 and 0.66.
- Support is the number of actual occurrences of the class in the specified dataset.

L. Conclusion

Decision tree classifier and logistic regression models were able to perform well with accuracy of 77.92% and 81.82% respectively. However, the logistic regression performed better than the decision tree and therefore it is a better classification model for this dataset.