Diabetes early detection & Prevention using Decision Tree and KNN

Table of Contents

Business Problem

This report deals with the early detection and prevention of diabetes dataset, the primary business problem to detect the diabetes under different parameters, these are pregnancies indicates the number of times that a person is pregnant because these dataset contains the information related to the female patients, Glucose: Glucose concentration in a tolerance test in 2 hours, Blood Pressure: it is measure in mm Hg, Skin thickness, thickness related to the triceps in mm, Insulin: measured 2-hour insulin in mu U/ml, BMI, Age in years and outcome as 0 and 1.

Goal

The goal of this analysis is to analyse the given data and helps the business in early detection of diabetes of female patients. We address this goal by two models' procedure KNN and Decision Trees, analyze the given data and perform the prediction using both the models and come up with the best model with best setting that work to predict the diabetes.

Feature Selection

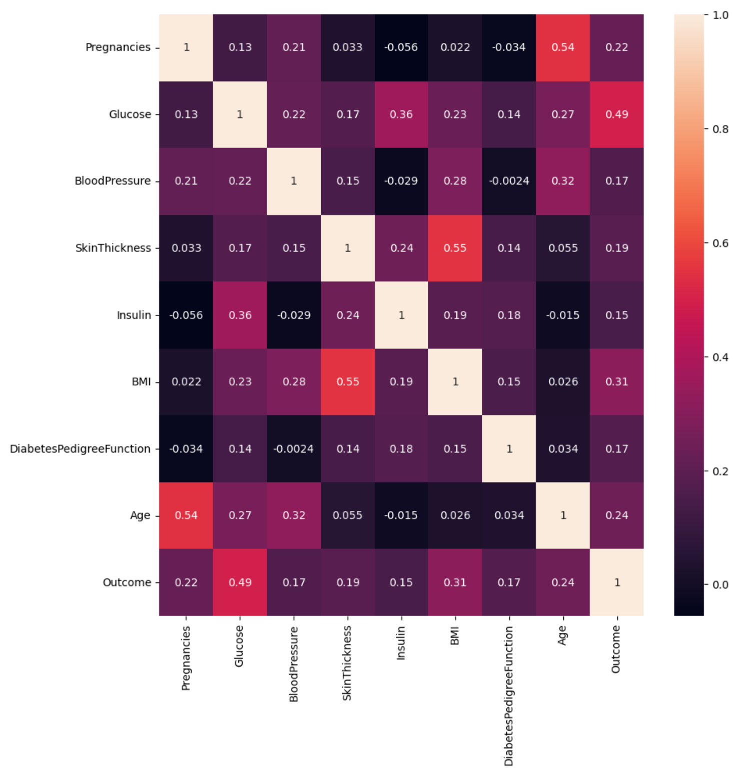
The dataset contains 768 rows and 9 columns, each column indicates the feature, and 9 features are important to predict the diabetes. Let us explore how these 9 features are very important, and we can also support the statement by finding the correlation between them.

- Pregnancies: Since this dataset contains the information related to the female patients, Pregnancy can influence glucose levels and insulin.
- Glucose: This Glucose has relevance since it is directly related to the diabetes and when you look at the correlation matric glucose is highly related to the outcome.

- Blood Pressure: The condition of blood pressure has potential significance on the diabetes.
- Skin Thickness: Skin thickness can be an indicator of bodyfat, and this has some relation with insulin and glucose level when you look at the correlation matrix.
- Insulin: Insulin is the crucial indicator as the diabetes can affect the insulin levels.
- BMI: BMI indicates body fat and can be helpful to understand the body type of the diabetes patients.
- Diabetes pedigree function: Identifies a person's genetic tendency to develop diabetes.
- Age: Age gives us the better picture of the patient’s body condition and diabetes risk.
- Outcome: Outcome is the target variable.

Fig 1:

Correlation matrix of features



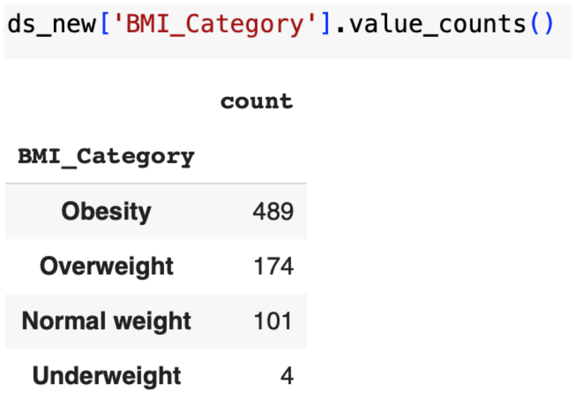
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1	0.13	0.21	0.033	-0.056	0.022	-0.034	0.54	0.22
Glucose	0.13	1	0.22	0.17	0.36	0.23	0.14	0.27	0.49
BloodPressure	0.21	0.22	1	0.15	-0.029	0.28	-0.0024	0.32	0.17
SkinThickness	0.033	0.17	0.15	1	0.24	0.55	0.14	0.055	0.19
Insulin	-0.056	0.36	-0.029	0.24	1	0.19	0.18	-0.015	0.15
BMI	0.022	0.23	0.28	0.55	0.19	1	0.15	0.026	0.31
DiabetesPedigreeFunction	-0.034	0.14	-0.0024	0.14	0.18	0.15	1	0.034	0.17
Age	0.54	0.27	0.32	0.055	-0.015	0.026	0.034	1	0.24
Outcome	0.22	0.49	0.17	0.19	0.15	0.31	0.17	0.24	1

Derived features:

- BMI Categories: Categorizing BMI into groups like underweight, Normal weight, Overweight and Obesity provides a better picture than the continuous values. With this category it is easily helps us to understand who are at higher risk of diabetes.

Fig2:

BMI Category count



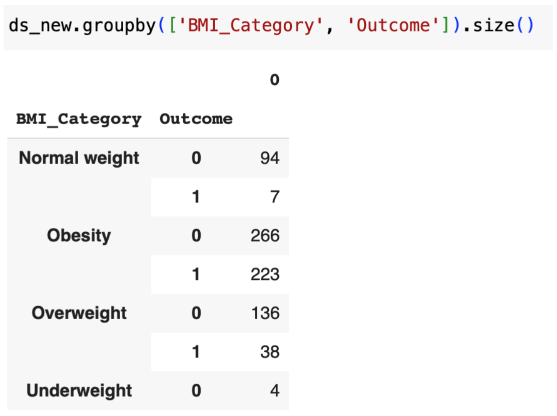
```
ds_new['BMI_Category'].value_counts()
```

BMI_Category	count
Obesity	489
Overweight	174
Normal weight	101
Underweight	4

This derived BMI category helps us to understand that most of dataset information contains patients with obesity and Overweight.

Fig 3:

BMI category grouped with the outcome



```
ds_new.groupby(['BMI_Category', 'Outcome']).size()
```

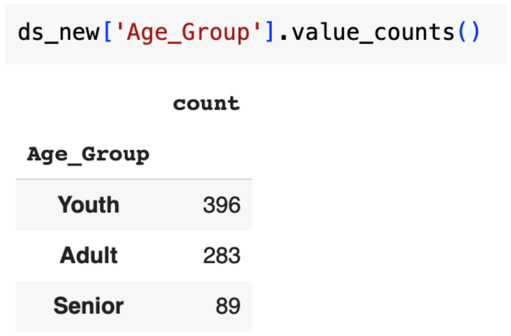
BMI_Category	Outcome	0
Normal weight	0	94
	1	7
Obesity	0	266
	1	223
Overweight	0	136
	1	38
Underweight	0	4

Further exploration of this gives us maximum diabetes patients are obesity and overweight. Surprisingly underweight patients have no diabetes and may requires more data to support this.

- Age group: Group age into Youth, Adult and Seniors helps us to understand the effect of diabetes on age and helps in taking care in early stages.

Fig 4:

Age group count from the dataset



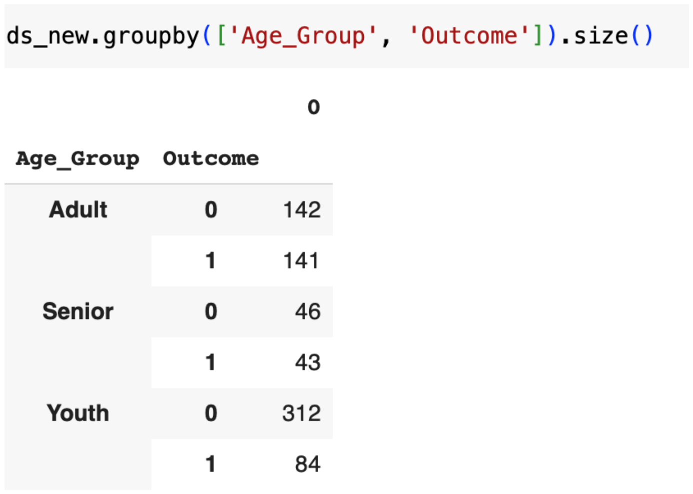
```
ds_new['Age_Group'].value_counts()
```

Age_Group	count
Youth	396
Adult	283
Senior	89

Age group reveals that dataset contains more Youth and Adults and Seniors.

Fig 5:

Outcomes with respect to age groups



```
ds_new.groupby(['Age_Group', 'Outcome']).size()
```

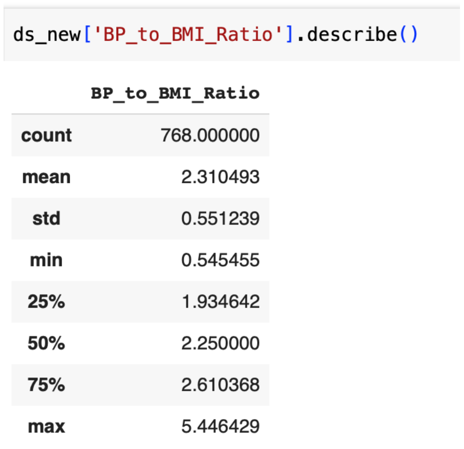
Age_Group	Outcome	0
Adult	0	142
	1	141
Senior	0	46
	1	43
Youth	0	312
	1	84

From this feature it is known that adult's risk of diabetes is high and from this dataset half are affected with diabetes and half of them are not. Most of the Youth are likely not affected with diabetes.

- Blood pressure to BMI Ratio: This ratio could indicate the relative impact of body mass on blood pressure. Most of the time, metabolic syndrome, which is a step towards diabetes, is marked by high blood pressure and a high body mass index.

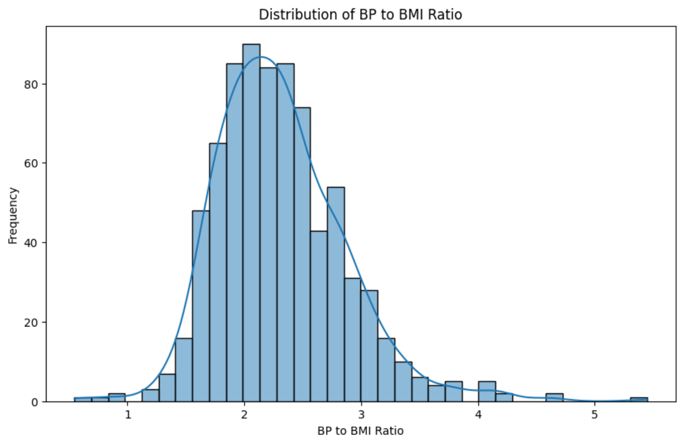
Fig 6:

Understanding BP to BMI ratio feature distribution



```
ds_new['BP_to_BMI_Ratio'].describe()
```

	BP_to_BMI_Ratio
count	768.000000
mean	2.310493
std	0.551239
min	0.545455
25%	1.934642
50%	2.250000
75%	2.610368
max	5.446429



This feature has an average value of 2.31, with most people having a ratio between 1.93 and 2.61. If the percentage is low, it means that the BMI is higher than the blood pressure, which may indicate that the person is overweight or obese without having high blood pressure. A higher ratio, on the other hand, means that Blood Pressure is significantly higher than BMI. This feature helps to find people who are more likely to get diabetes based on the balance between their blood pressure and body weight.

Data Quality Resolution

The given dataset contains missing values.

Fig 7:

Features with missing values

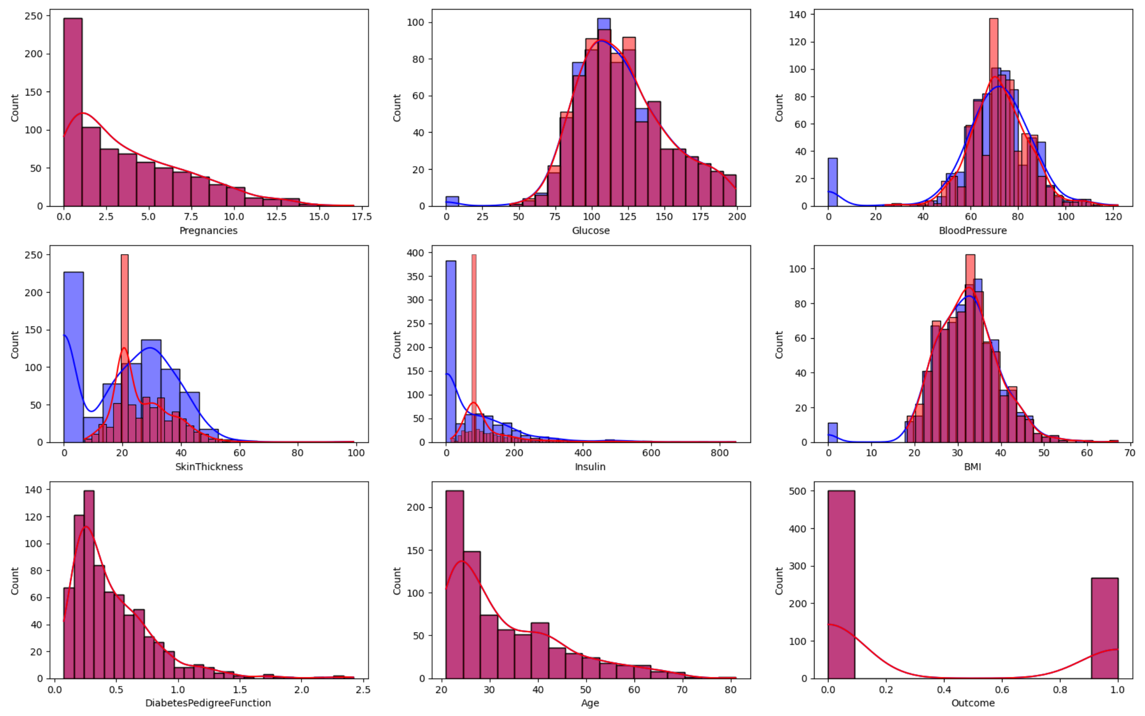
```
Pregnancies: 111
Glucose: 5
BloodPressure: 35
SkinThickness: 227
Insulin: 374
BMI: 11
DiabetesPedigreeFunction: 0
Age: 0
```

To handle the missing values, we do the following steps.

1. Filling with mean values: First let us fill the missing values in the dataset with the mean of their respected columns and identify the distribution.

   Fig 8:

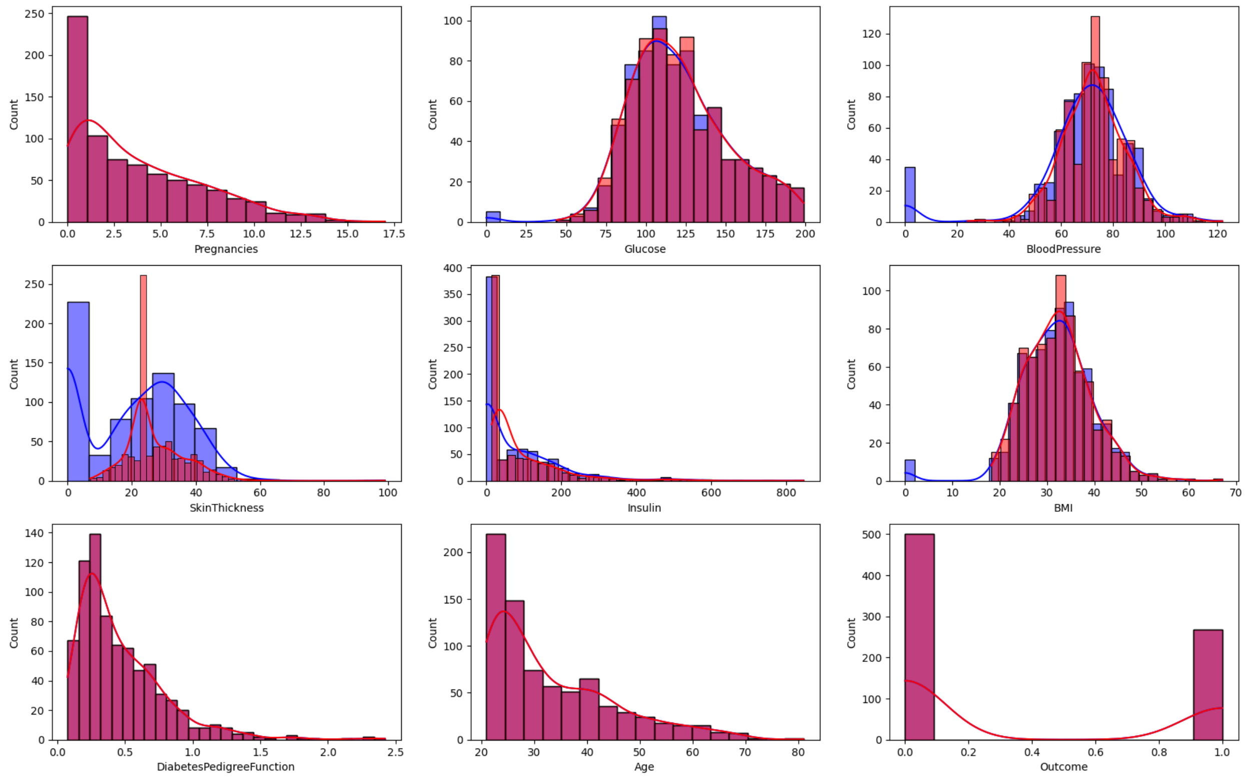
   Distribution of dataset after filling the missing values with mean



2. Filling with Median: Filling the missing values with the median and understand the distribution before and after filling.

Fig 9:

Distribution of dataset after filling the missing values with median

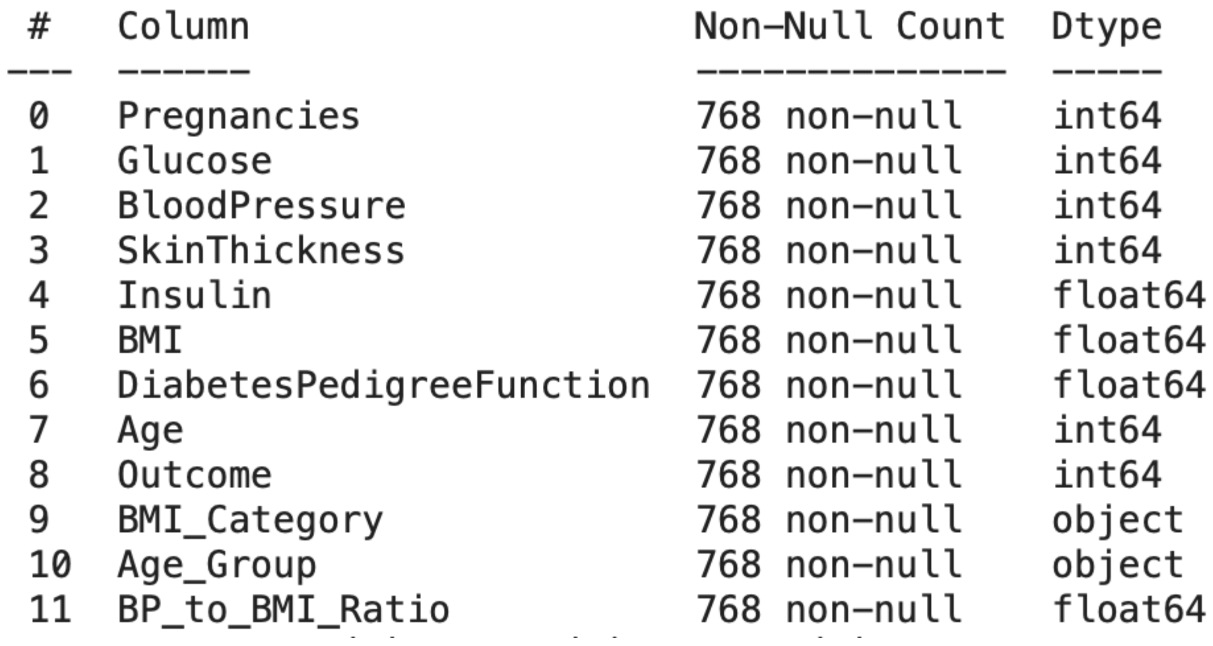


Looking at both the distributions, filling the missing value with the median value fits with the original distribution and we choose median value to fill the missing values in the dataset.

After handling the missing values and adding the derived features to the dataset, the follow fig represents the final ABT with 11 features.

Fig 10:

Final ABT table features



```
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    float64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
 9   BMI_Category              768 non-null    object
 10  Age_Group                 768 non-null    object
 11  BP_to_BMI_Ratio           768 non-null    float64
```

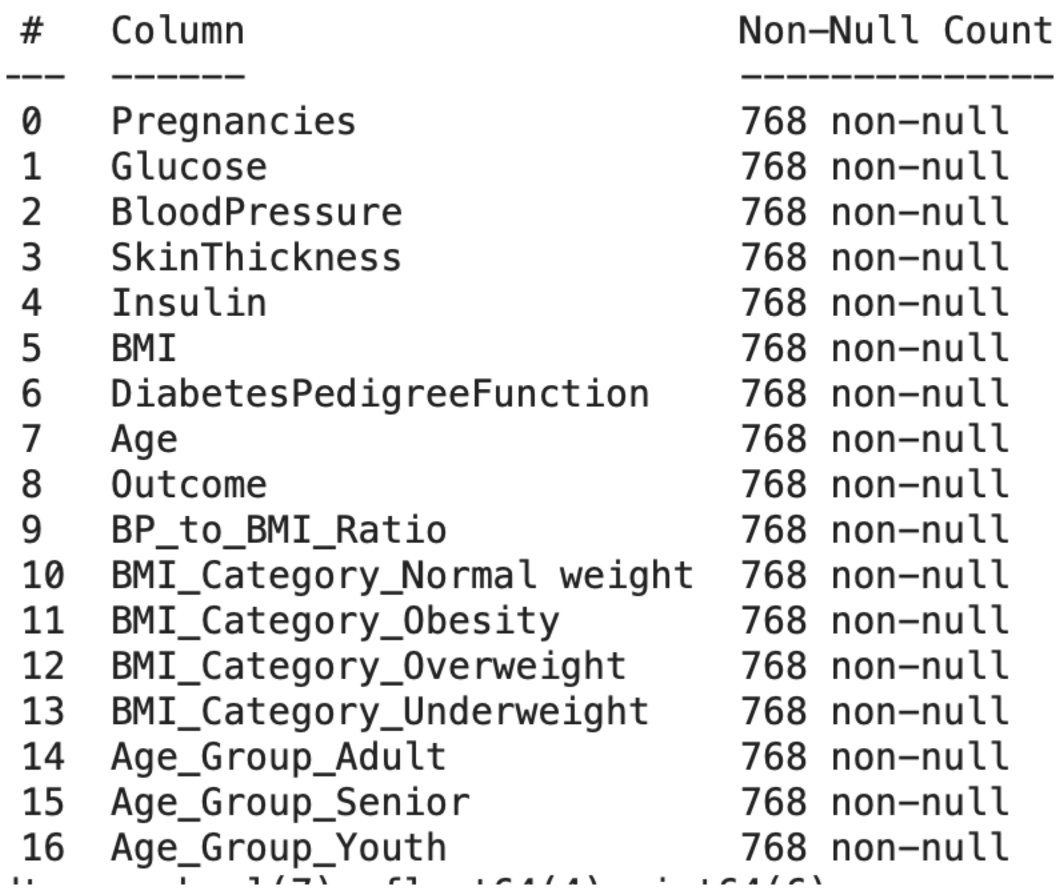
Data Processing

a. One Hot Encoding:

The derived features “BMI_category” and “Age” contains categorical data, using one hot encoding let us convert the categorical features into numerical which helps in the analysis.

Fig 11:

ABT after one hot encoding

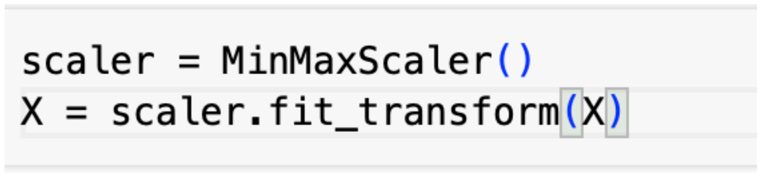


```
 #   Column                       Non-Null Count
---  ------                       --------------
 0   Pregnancies                  768 non-null
 1   Glucose                      768 non-null
 2   BloodPressure                768 non-null
 3   SkinThickness                768 non-null
 4   Insulin                      768 non-null
 5   BMI                          768 non-null
 6   DiabetesPedigreeFunction     768 non-null
 7   Age                          768 non-null
 8   Outcome                      768 non-null
 9   BP_to_BMI_Ratio              768 non-null
 10  BMI_Category_Normal weight  768 non-null
 11  BMI_Category_Obesity         768 non-null
 12  BMI_Category_Overweight      768 non-null
 13  BMI_Category_Underweight     768 non-null
 14  Age_Group_Adult              768 non-null
 15  Age_Group_Senior             768 non-null
 16  Age_Group_Youth              768 non-null
```

b. **Normalization:** To Normalize the data, minmax scalar from sklearn.preprocessing is used to scale the data set between 0-1.

Fig 12:

MinMax Scalar



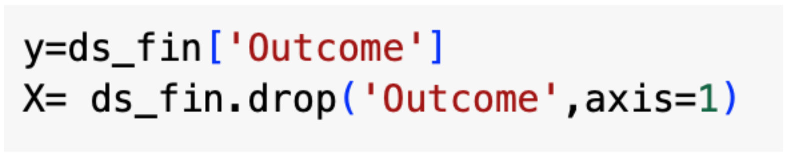
```
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
```

Data Splitting

After the completing of data processing, the target feature is assigned to "y" and the independent variables are assigned to "X" as showing in the fig 13.

Fig 13:

Assigning the features to X and y

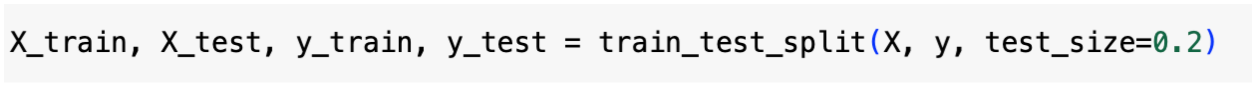


```
y=ds_fin['Outcome']
X= ds_fin.drop('Outcome',axis=1)
```

After assigning the features to X and y, we split the features into training and testing with the help of "train_test_split" from sklearn.model_selection with train and test size of 80 and 20.

Fig 14:

Splitting the data set into train and test



```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

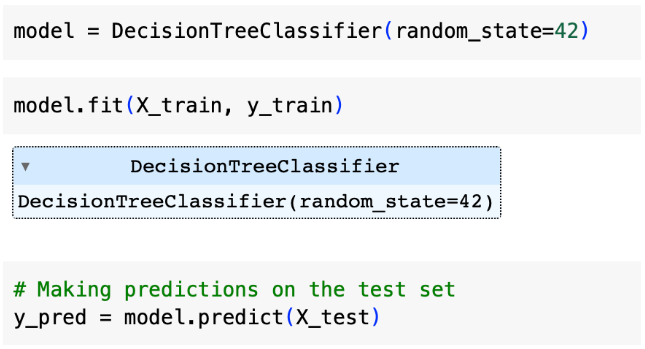
Model Training

After the completion of splitting the data set, we train the model with two models

a. Decision Tree: First, we train the model with Decision Tree Classifier, we import the model from sklearn. tree. We use Decision Tree Classifier because the target feature is not continuous, if the target feature is continuous, we use Decision Tree Regressor. After importing the model fit X_train and y_train with the model and y_pred is used to predict the model as shown in the fig 15.

Fig 15:

Importing the Decision Tree Classifier model



```
model = DecisionTreeClassifier(random_state=42)
```

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

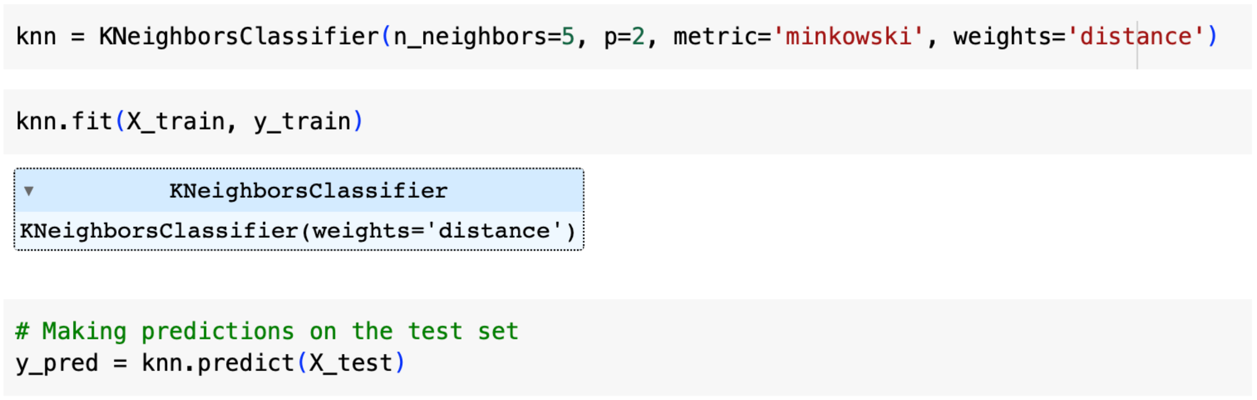
```
▾          DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

```
# Making predictions on the test set
y_pred = model.predict(X_test)
```

b. KNN: We also train our model using KNN, we import “KNeighboursClassifier” from sklearn. neighbours and fit X_train and y_train using knn.fit as shown in the fig 16.

Fig 16:

Importing KNeighbour Classifier



```
knn = KNeighborsClassifier(n_neighbors=5, p=2, metric='minkowski', weights='distance')
```

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

```
▾          KNeighborsClassifier
KNeighborsClassifier(weights='distance')
```

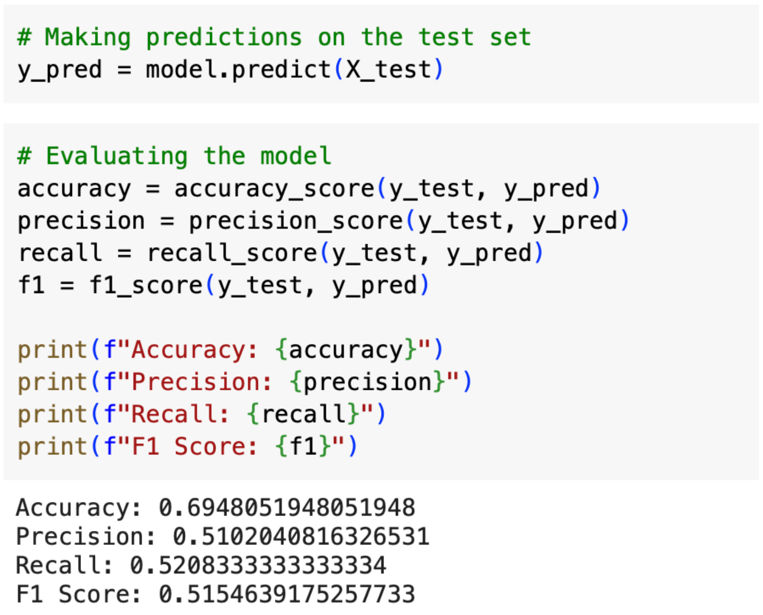
```
# Making predictions on the test set
y_pred = knn.predict(X_test)
```

Model Testing and Evaluation

a. Decision Tree: The model is trained and evaluated with the help of accuracy, precision, recall and f1 score from sklearn. metrics as shown in the fig 17.

Fig 17:

Evaluating the model



```
# Making predictions on the test set
y_pred = model.predict(X_test)
```

```
# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
```

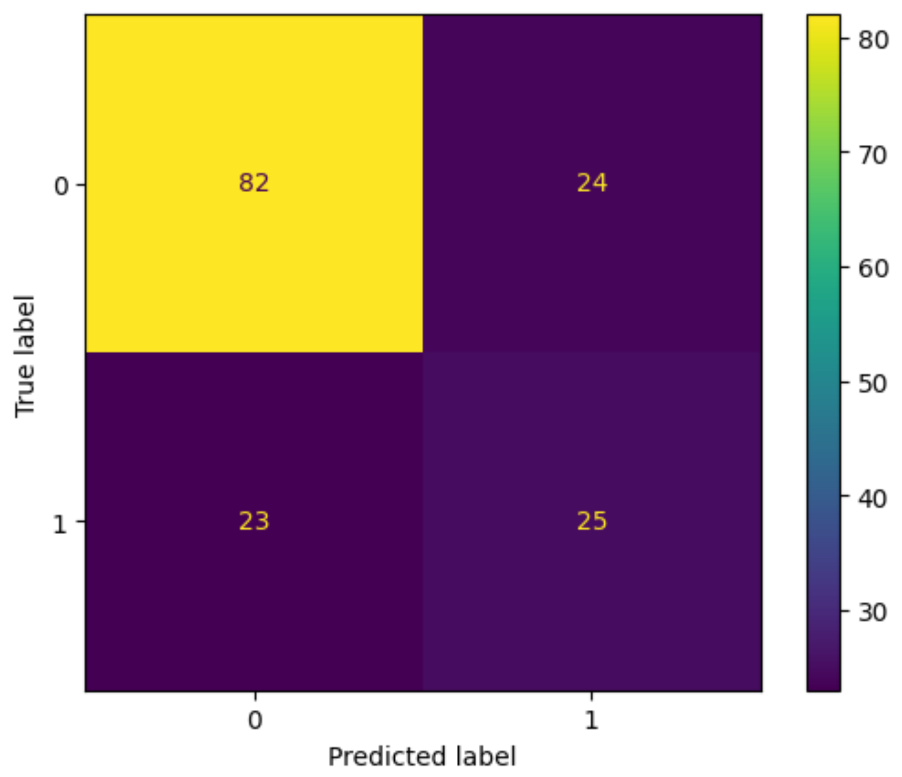
```
Accuracy: 0.6948051948051948
Precision: 0.5102040816326531
Recall: 0.5208333333333334
F1 Score: 0.5154639175257733
```

- The model accurately identified about 69.48% of the test samples. While this suggests a generally good performance from the base line, accuracy alone might be misleading.
- A precision of 51.02% indicates that when the model predicts a positive outcome, it is correct approximately fifty percent. This shows that the model produces a moderate number of false positives. Precision improves when false positives decrease.
- A recall of 52.08% indicates that the model accurately detected slightly more than half of the true positive cases. This suggests the model may be missing some positive cases. Recall improves when false negatives decrease.
- F1 score of 51.55% indicates a compromise between precision and recall, indicating that the model needs work in achieving a balance between detecting positive cases and accurately predicting them.

Confusion Matrix:

Fig 18:

Confusion matrix



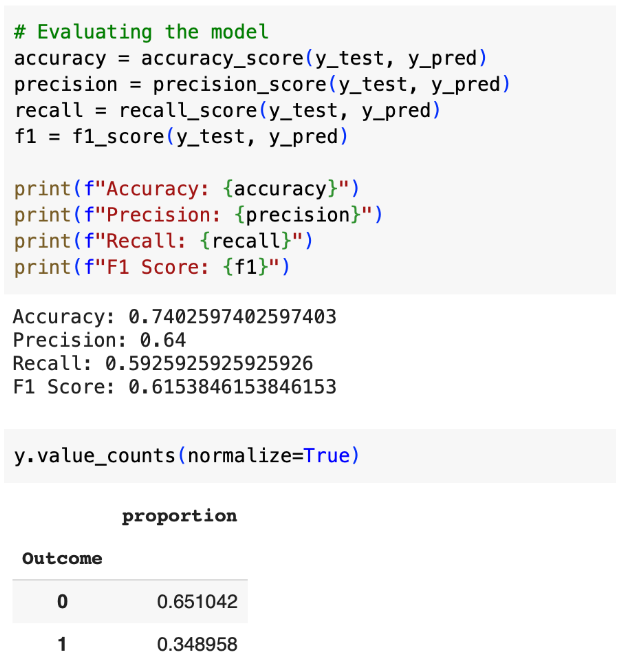
The confusion matrix tells that the model predicted

- True Positives (25): Correctly identified positive instances.
- True Negatives (82): Correctly identified negative instances.
- False Positives (24): Incorrectly identified negative instances as positive.
- False Negatives (23): Incorrectly identified positive instances as negative.

b. KNN Model: The model is trained and evaluated with the help of accuracy, precision, recall and f1 score from sklearn. metrics as shown in the fig 19.

Fig 19:

Evaluating model



```
# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
```

```
Accuracy: 0.7402597402597403
Precision: 0.64
Recall: 0.5925925925925926
F1 Score: 0.6153846153846153
```

```
y.value_counts(normalize=True)
```

	proportion
Outcome	
0	0.651042
1	0.348958

- A 74.0% accuracy indicates that the model demonstrates good accuracy from the base line accuracy of 65%. However more tuning and different parameter changes are required to increase the accuracy of the model.
- A precision of 64% means that when the model predicts a positive class, it is correct about 64% of the time. This indicates that the model has a moderate level of correctness in its predictions.
- A recall rate of 59.3% signifies that the model can correctly identify around 59.3% of all True positive occurrences. This is a metric that quantifies the model's capacity to accurately identify and include as many positive outcomes as possible.
- A score of 61.5% indicates a moderate equilibrium between precision and recall, which represents the whole performance by considering both false positives and false negatives.

Hyperparameter Tuning

a. Decision Tree: Adjusting different settings for the model to get the best outcome.

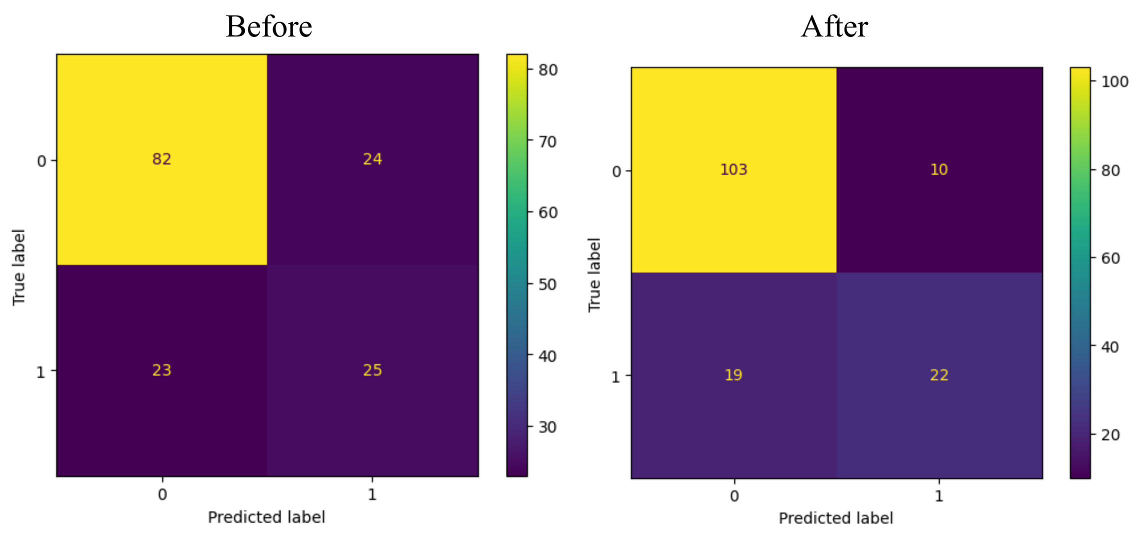
Depths	Depth = 3 Accuracy: 0.7272727272727273 Precision: 0.6444444444444445 Recall: 0.5272727272727272 F1 Score: 0.58 Depth = 4 Accuracy: 0.7142857142857143 Precision: 0.5846153846153846 Recall: 0.6909090909090909 F1 Score: 0.6333333333333333 Depth = 5 Accuracy: 0.6883116883116883 Precision: 0.5573770491803278 Recall: 0.6181818181818182 F1 Score: 0.5862068965517241 Depth = 6 Accuracy: 0.6948051948051948 Precision: 0.5769230769230769 Recall: 0.5454545454545454 F1 Score: 0.5607476635514018 Depth = 7 Accuracy: 0.6558441558441559 Precision: 0.5166666666666667 Recall: 0.5636363636363636 F1 Score: 0.5391304347826087
criterion= 'entropy'	Depth = 3 and Criterion= entropy Accuracy: 0.8116883116883117 Precision: 0.6875 Recall: 0.5365853658536586 F1 Score: 0.6027397260273972
Criterion= 'Gini'	Depth = 3 and Criterion Gini Accuracy: 0.7142857142857143 Precision: 0.5675675675675675 Recall: 0.42857142857142855 F1 Score: 0.4883720930232558

Observing the different setting and their accuracy the best setting for decision tree is found to be depth=3 with criterion “entropy’ with an accuracy of 81%.

Confusion Matrix:

Fig 20:

Confusion matrix before and after tuning



The comparison of the confusion matrices before and after tuning indicates an improvement in the performance of the model. The initial model comprised 82 occurrences correctly identified as negatives and 25 instances correctly identified as positives, along with 24 instances incorrectly identified as positives and 23 instances incorrectly identified as negatives. Upon tuning different settings, the number of true negatives rose to 103, while true positives experienced a tiny decline to 22. However, false positives were dramatically reduced to 10, which suggests that the model is now making fewer wrong predictions for the negative class, thus improving its overall accuracy.

b. KNN:

neighbors=1-20,
p=2,
metric='Manhattan',
weights= distance
	# of neighbours: 1
	Accuracy: 0.7077922077922078
	# of neighbours: 2
	Accuracy: 0.7207792207792207
	# of neighbours: 3
	Accuracy: 0.7467532467532467
	# of neighbours: 4
	Accuracy: 0.7532467532467533
	# of neighbours: 5
	Accuracy: 0.7662337662337663
	# of neighbours: 6
	Accuracy: 0.7272727272727273
	# of neighbours: 7
	Accuracy: 0.7337662337662337
	# of neighbours: 8
	Accuracy: 0.7402597402597403
	# of neighbours: 9
	Accuracy: 0.7467532467532467
	# of neighbours: 10
	Accuracy: 0.7467532467532467
	# of neighbours: 11
	Accuracy: 0.7467532467532467
	# of neighbours: 12
	Accuracy: 0.7467532467532467
	# of neighbours: 13
	Accuracy: 0.7597402597402597
	# of neighbours: 14
	Accuracy: 0.7662337662337663
	# of neighbours: 15
	Accuracy: 0.7597402597402597
	# of neighbours: 16
	Accuracy: 0.7467532467532467
	# of neighbours: 17
	Accuracy: 0.7467532467532467
	# of neighbours: 18
	Accuracy: 0.7467532467532467
	# of neighbours: 19
	Accuracy: 0.7662337662337663
	# of neighbours: 20
	Accuracy: 0.7597402597402597

neighbors=1-20,
p=2,
metric='Manhattan',
weights= 'uniform'
	# of neighbours: 1
	Accuracy: 0.7077922077922078
	# of neighbours: 2
	Accuracy: 0.7207792207792207

	# of neighbours: 3 Accuracy: 0.7467532467532467 # of neighbours: 4 Accuracy: 0.7532467532467533 # of neighbours: 5 Accuracy: 0.7662337662337663 # of neighbours: 6 Accuracy: 0.7272727272727273 # of neighbours: 7 Accuracy: 0.7337662337662337 # of neighbours: 8 Accuracy: 0.7402597402597403 # of neighbours: 9 Accuracy: 0.7467532467532467 # of neighbours: 10 Accuracy: 0.7467532467532467 # of neighbours: 11 Accuracy: 0.7467532467532467 # of neighbours: 12 Accuracy: 0.7467532467532467 # of neighbours: 13 Accuracy: 0.7597402597402597 # of neighbours: 14 Accuracy: 0.7662337662337663 # of neighbours: 15 Accuracy: 0.7597402597402597 # of neighbours: 16 Accuracy: 0.7467532467532467 # of neighbours: 17 Accuracy: 0.7467532467532467 # of neighbours: 18 Accuracy: 0.7467532467532467 # of neighbours: 19 Accuracy: 0.7662337662337663 # of neighbours: 20 Accuracy: 0.7597402597402597

neighbors=1-20,
p=2,
metric=minkowski,
weights= distance
	# of neighbours: 1
	Accuracy: 0.6883116883116883
	# of neighbours: 2
	Accuracy: 0.6883116883116883
	# of neighbours: 3
	Accuracy: 0.7142857142857143
	# of neighbours: 4
	Accuracy: 0.7142857142857143
	# of neighbours: 5

	Accuracy: 0.7402597402597403
	# of neighbours: 6
	Accuracy: 0.7077922077922078
	# of neighbours: 7
	Accuracy: 0.7272727272727273
	# of neighbours: 8
	Accuracy: 0.7402597402597403
	# of neighbours: 9
	Accuracy: 0.7597402597402597
	# of neighbours: 10
	Accuracy: 0.7597402597402597
	# of neighbours: 11
	Accuracy: 0.7727272727272727
	# of neighbours: 12
	Accuracy: 0.7532467532467533
	# of neighbours: 13
	Accuracy: 0.7727272727272727
	# of neighbours: 14
	Accuracy: 0.7597402597402597
	# of neighbours: 15
	Accuracy: 0.7597402597402597
	# of neighbours: 16
	Accuracy: 0.7662337662337663
	# of neighbours: 17
	Accuracy: 0.7532467532467533
	# of neighbours: 18
	Accuracy: 0.7597402597402597
	# of neighbours: 19
	Accuracy: 0.7662337662337663
	# of neighbours: 20
	Accuracy: 0.7597402597402597
neighbors=1-20,	# of neighbours: 1
p=2,	Accuracy: 0.6883116883116883
metric=minkowski,	# of neighbours: 2
weights= uniform	Accuracy: 0.7272727272727273
	# of neighbours: 3
	Accuracy: 0.7077922077922078
	# of neighbours: 4
	Accuracy: 0.7402597402597403
	# of neighbours: 5
	Accuracy: 0.7402597402597403
	# of neighbours: 6
	Accuracy: 0.7142857142857143
	# of neighbours: 7
	Accuracy: 0.7142857142857143
	# of neighbours: 8

	Accuracy: 0.7272727272727273
	# of neighbours: 9
	Accuracy: 0.7597402597402597
	# of neighbours: 10
	Accuracy: 0.7727272727272727
	# of neighbours: 11
	Accuracy: 0.7727272727272727
	# of neighbours: 12
	Accuracy: 0.7662337662337663
	# of neighbours: 13
	Accuracy: 0.7727272727272727
	# of neighbours: 14
	Accuracy: 0.7532467532467533
	# of neighbours: 15
	Accuracy: 0.7792207792207793
	# of neighbours: 16
	Accuracy: 0.7467532467532467
	# of neighbours: 17
	Accuracy: 0.7402597402597403
	# of neighbours: 18
	Accuracy: 0.7207792207792207
	# of neighbours: 19
	Accuracy: 0.7337662337662337
	# of neighbours: 20
	Accuracy: 0.7402597402597403

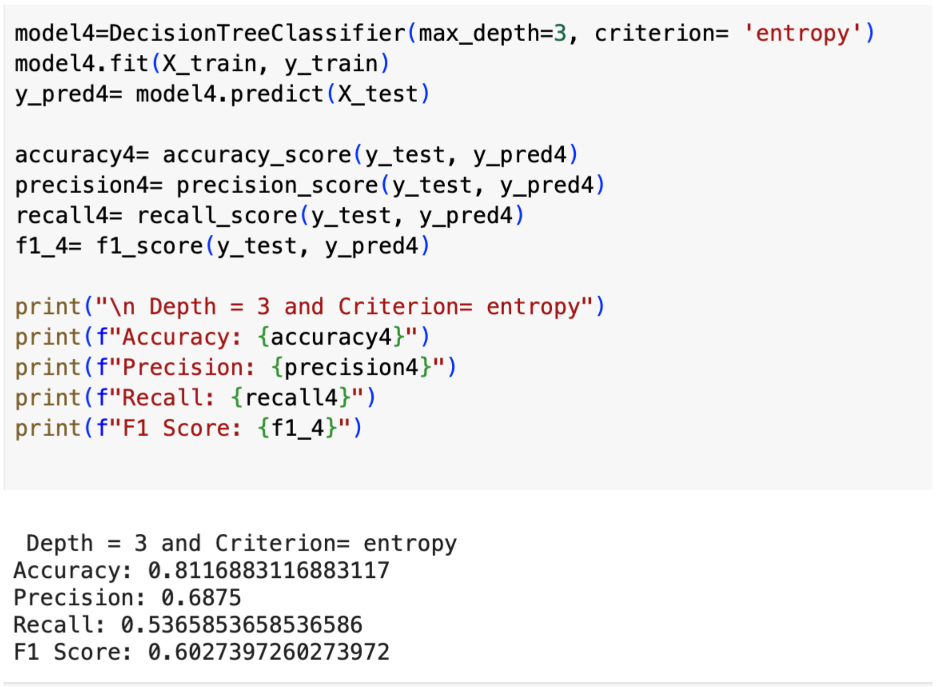
The best accuracy is found to be 77.92% at 15 neighbours with p=2, metric= minkowski and weights are uniform.

Best Model Settings

a. Decision Tree: The best model setting for decision tress is as at depth 3 and criterion is entropy, and the accuracy is found to be 81.1% as shown in the fig 21.

Fig 21:

Best settings for Decision Tree

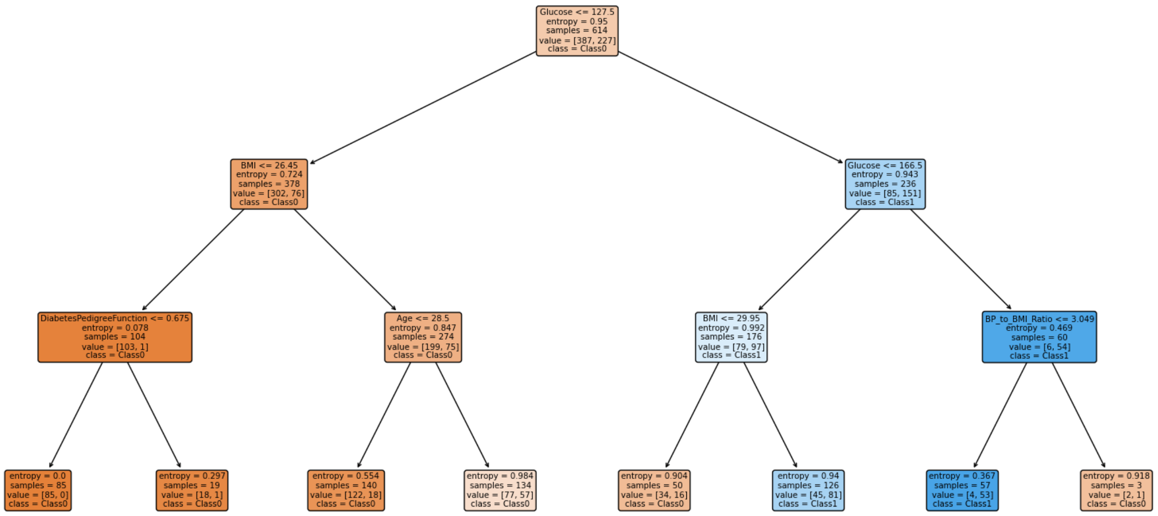


```
model4=DecisionTreeClassifier(max_depth=3, criterion= 'entropy')
model4.fit(X_train, y_train)
y_pred4= model4.predict(X_test)

accuracy4= accuracy_score(y_test, y_pred4)
precision4= precision_score(y_test, y_pred4)
recall4= recall_score(y_test, y_pred4)
f1_4= f1_score(y_test, y_pred4)

print("\n Depth = 3 and Criterion= entropy")
print(f"Accuracy: {accuracy4}")
print(f"Precision: {precision4}")
print(f"Recall: {recall4}")
print(f"F1 Score: {f1_4}")
```

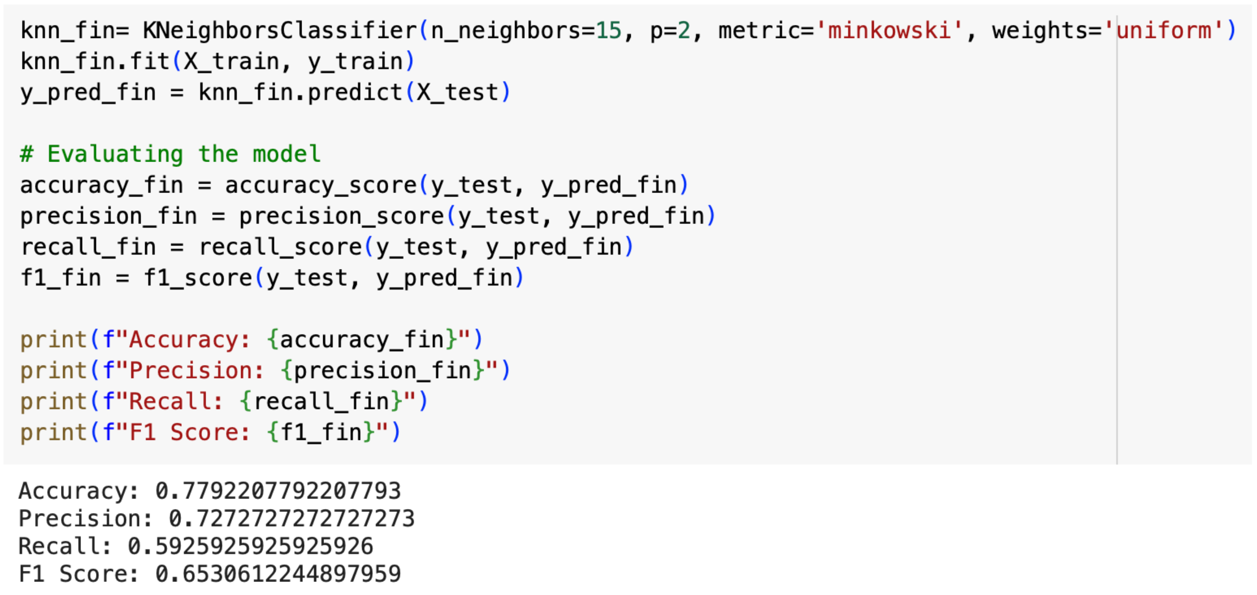
```
 Depth = 3 and Criterion= entropy
Accuracy: 0.8116883116883117
Precision: 0.6875
Recall: 0.5365853658536586
F1 Score: 0.6027397260273972
```



b. KNN: The best model setting for decision tress is as at neighbours=15, metric= minkowski and weights = uniform and the accuracy is found to be 77.92% as shown in the fig 22.

Fig 22:

Best settings for KNN



```
knn_fin= KNeighborsClassifier(n_neighbors=15, p=2, metric='minkowski', weights='uniform')
knn_fin.fit(X_train, y_train)
y_pred_fin = knn_fin.predict(X_test)

# Evaluating the model
accuracy_fin = accuracy_score(y_test, y_pred_fin)
precision_fin = precision_score(y_test, y_pred_fin)
recall_fin = recall_score(y_test, y_pred_fin)
f1_fin = f1_score(y_test, y_pred_fin)

print(f"Accuracy: {accuracy_fin}")
print(f"Precision: {precision_fin}")
print(f"Recall: {recall_fin}")
print(f"F1 Score: {f1_fin}")
```

```
Accuracy: 0.7792207792207793
Precision: 0.7272727272727273
Recall: 0.5925925925925926
F1 Score: 0.6530612244897959
```

Model Comparison

Decision Tree	**KNN**
Accuracy: 0.8116883116883117 Precision: 0.6875 Recall: 0.5365853658536586 F1 Score: 0.6027397260273972	Accuracy: 0.7792207792207793 Precision: 0.7272727272727273 Recall: 0.5925925925925926 F1 Score: 0.6530612244897959

With its greater Precision, Recall, and F1 Score, KNN is the better option for this problem. According to these parameters, KNN reliably and effectively identifies positive situations. While Decision Trees is slightly better in total accuracy, they may not be as dependable when it comes to correctly selecting the positive situations due to their lower Precision, Recall, and F1 Score.

Dataset Link

This is the dataset that I've used for the assignment.

https://www.kaggle.com/datasets/mathchi/diabetes-data-set/data

Colab link

This is the link for KNN model:

https://colab.research.google.com/drive/1AF1NfOBQAFdkzyT5BQK7CcAuB3PiMXG0?usp=sh
aring

This is the link for Decision Tree model:

https://colab.research.google.com/drive/11-
ABKY0H1ZGuvB0XVtgPCl6kypaD8edh?usp=sharing