Project report

RA2011042010092-B.VENKATA VISWANATH REDDY

RA2011042010121-GOPINADH

Dataset: https://www.kaggle.com/johndasilva/diabetes

Diabetes Prediction Using Machine Learning

Abstract

Diabetes Mellitus is among critical diseases and lots of people are suffering from this disease. Age, obesity, lack of exercise, hereditary diabetes, living style, bad diet, high blood pressure, etc. can cause Diabetes Mellitus. People having diabetes have high risk of diseases like heart disease, kidney disease, stroke, eye problem, nerve damage, etc. Current practice in hospital is to collect required information for diabetes diagnosis through various tests and appropriate treatment is provided based on diagnosis. Big Data Analytics plays an significant role in healthcare industries. Healthcare industries have large volume databases. Using big data analytics one can study huge datasets and find hidden information, hidden patterns to discover knowledge from the data and predict outcomes accordingly. In existing method, the classification and prediction accuracy is not so high. In this paper, we have proposed a diabetes prediction model for better classification of diabetes which includes few external factors responsible for diabetes

along with regular factors like Glucose, BMI, Age, Insulin, etc. Classification accuracy is boosted with new dataset compared to existing dataset. Further with imposed a pipeline model for diabetes prediction intended towards improving the accuracy of classification.

1. Data analysis: Here one will get to know about how the data analysis part is done in a data science life cycle.
2. Exploratory data analysis: EDA is one of the most important steps in the data science project life cycle and here one will need to know that how to make inferences from the visualizations and data analysis
3. Model building: Here we will be using 4 ML models and then we will choose the best performing model.

Importing Libraries

```
In [1]: import numpy as np
        import pandas as pd
```

Here we will be reading the dataset which is in the CSV format

```
In [2]: df = pd.read_csv('diabetes.csv')
```

```
In [3]: df
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	2	138	62	35	0	33.6	0.127	47	1
1	0	84	82	31	125	38.2	0.233	23	0
2	0	145	0	0	0	44.2	0.630	31	1
3	0	135	68	42	250	42.3	0.365	24	1
4	1	139	62	41	480	40.7	0.536	21	0
...	...	...	...	...	...	...	...	...	...
1995	2	75	64	24	55	29.7	0.370	33	0
1996	8	179	72	42	130	32.7	0.719	36	1
1997	6	85	78	0	0	31.2	0.382	42	0
1998	0	129	110	46	130	67.1	0.319	26	1
1999	2	81	72	15	76	30.1	0.547	25	0

2000 rows × 9 columns

Exploratory Data Analysis (EDA)

Now let’ see that what are columns available in our dataset.

```
In [5]: df.columns
```

```
Out[5]: Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
               'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
              dtype='object')
```

To know more about the dataset

```
# Returns basic statistics on numeric columns
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Pregnancies	2000.0	3.70350	3.306063	0.000	1.000	3.000	6.000	17.00
Glucose	2000.0	121.18250	32.068636	0.000	99.000	117.000	141.000	199.00
BloodPressure	2000.0	69.14550	19.188315	0.000	63.500	72.000	80.000	122.00
SkinThickness	2000.0	20.93500	16.103243	0.000	0.000	23.000	32.000	110.00
Insulin	2000.0	80.25400	111.180534	0.000	0.000	40.000	130.000	744.00
BMI	2000.0	32.19300	8.149901	0.000	27.375	32.300	36.800	80.60
DiabetesPedigreeFunction	2000.0	0.47093	0.323553	0.078	0.244	0.376	0.624	2.42
Age	2000.0	33.09050	11.786423	21.000	24.000	29.000	40.000	81.00
Outcome	2000.0	0.34200	0.474498	0.000	0.000	0.000	1.000	1.00

Now let’s check that if our dataset have null values or not

```
In [11]: # Returns true for a column having null values, else false
         df.isnull().any()

Out[11]: Pregnancies                 False
         Glucose                     False
         BloodPressure               False
         SkinThickness               False
         Insulin                     False
         BMI                         False
         DiabetesPedigreeFunction    False
         Age                         False
         Outcome                     False
         dtype: bool
```

Here from the above code we first checked that is there any null values from the **IsNull()** function then we are going to take the sum of all those missing values from the **sum()** function and the inference we now get is that there are no missing values but that is actually not a true story as in **this particular dataset all the missing values were given the 0 as a value which is not good for the authenticity of the dataset.** Hence we will first **replace the 0 value with the NAN** value then start the imputation process.

```
In [15]: # Replacing the 0 values from ['Glucose','BloodPressure','SkinThickness','Insulin','BMI'] by NaN
         df_copy = df.copy(deep=True)
         df_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = df_copy[['Glucose','BloodPressure','SkinThickness','Insuli
         df_copy.isnull().sum()

Out[15]: Pregnancies       0
         Glucose          13
         BloodPressure    90
         SkinThickness   573
         Insulin         956
         BMI              28
         DPF               0
         Age               0
         Outcome           0
         dtype: int64
```

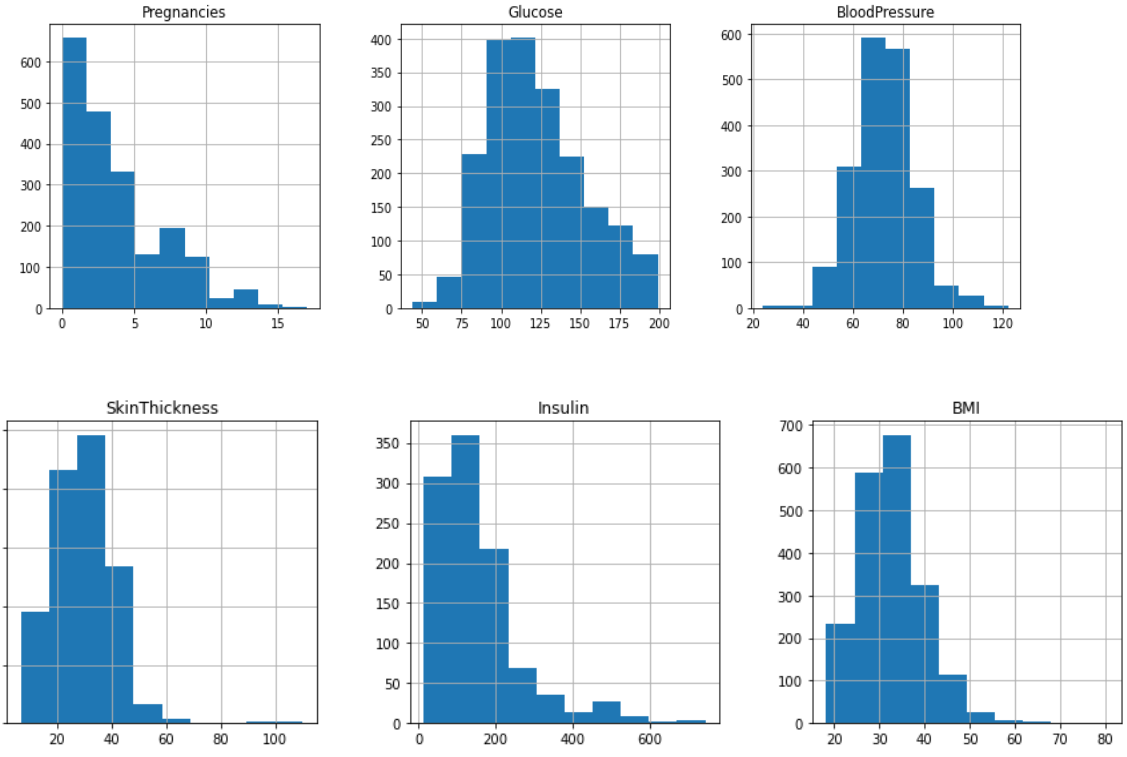
As mentioned above that now **we will be replacing the zeros with the NAN values** so that we can impute it later to maintain the

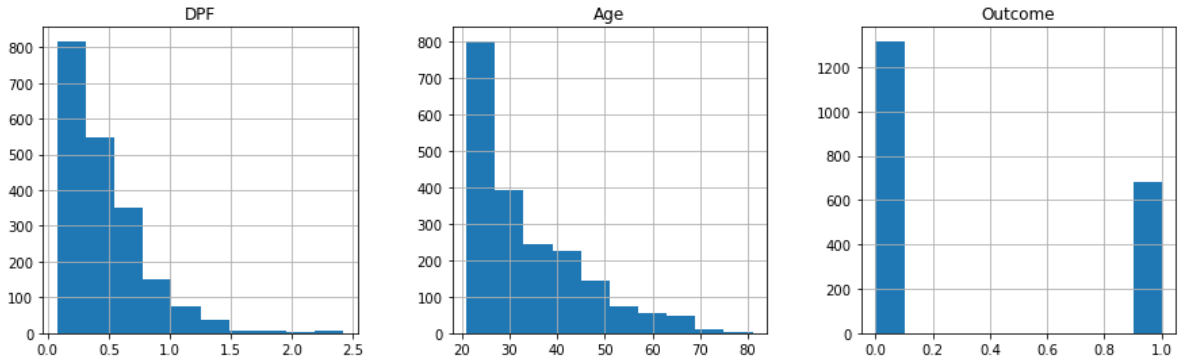
authenticity of the dataset as well as trying to have a better Imputation approach i.e **to apply mean values of each column to the null values of the respective columns.**

Data Visualization

Plotting the data distribution plots before removing null values

```
In [16]: # To fill these Nan values the data distribution needs to be understood
         # Plotting histogram of dataset before replacing NaN values
         p = df_copy.hist(figsize = (15,15))
```





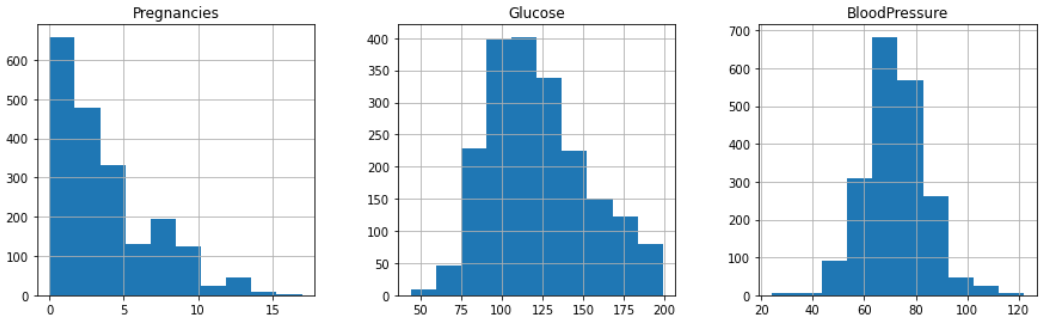
Inference: So here we have seen the distribution of each features whether it is dependent data or independent data and one thing which could always strike that **why do we need to see the distribution of data?** So the answer is simple it is the best way to start the analysis of the dataset as **it shows the occurrence of every kind of value in the graphical structure which in turn lets us know the range of the data.**

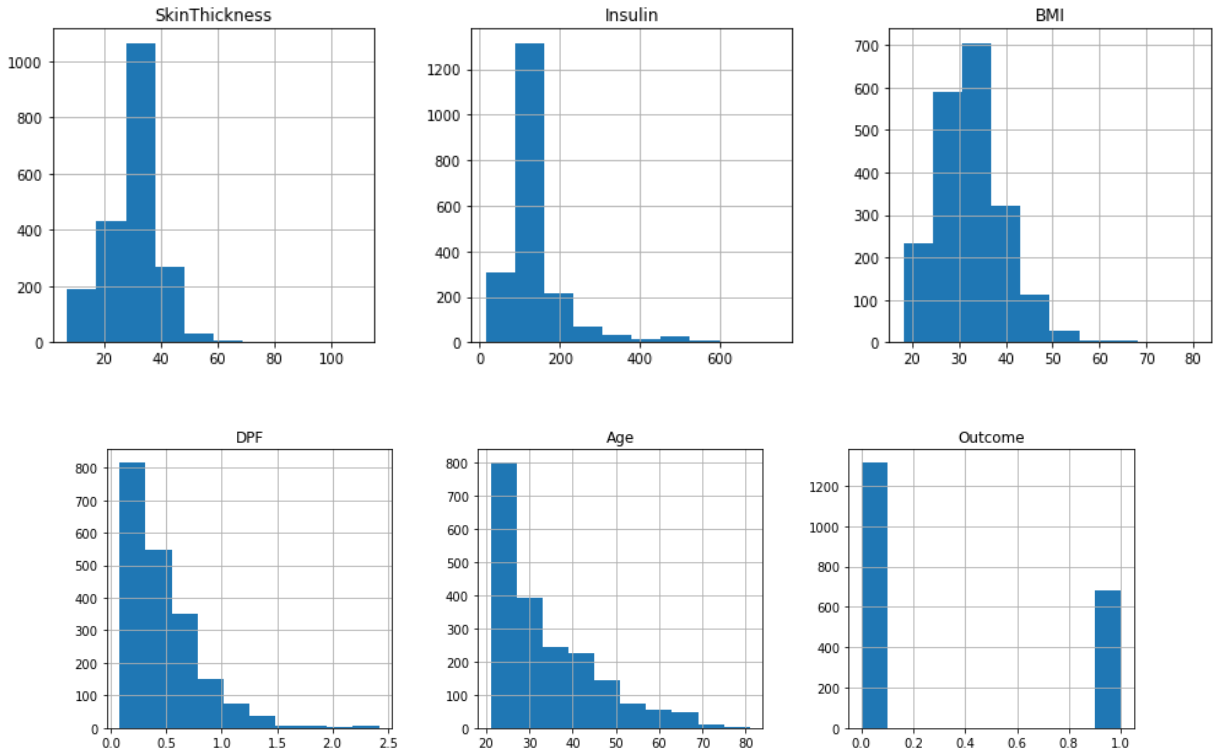
Now we will be imputing the mean value of the column to each missing value of that particular column.

In [17]:
```
# Replacing NaN value by mean, median depending upon distribution
df_copy['Glucose'].fillna(df_copy['Glucose'].mean(), inplace=True)
df_copy['BloodPressure'].fillna(df_copy['BloodPressure'].mean(), inplace=True)
df_copy['SkinThickness'].fillna(df_copy['SkinThickness'].median(), inplace=True)
df_copy['Insulin'].fillna(df_copy['Insulin'].median(), inplace=True)
df_copy['BMI'].fillna(df_copy['BMI'].median(), inplace=True)
```

Plotting the distributions after removing the NAN values.

In [18]:
```
# Plotting histogram of dataset after replacing NaN values
p = df_copy.hist(figsize=(15,15))
```





Now we will split the data into training and testing data using the train_ test_ split function

In [20]:
```
from sklearn.model_selection import train_test_split

X = df.drop(columns='Outcome')
y = df['Outcome']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=0)
print('X_train size: {}, X_test size: {}'.format(X_train.shape, X_test.shape))
```

```
X_train size: (1600, 8), X_test size: (400, 8)
```

Using GridSearchCV to find the best algorithm for this problem

In [22]:
```
# Using GridSearchCV to find the best algorithm for this problem
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import ShuffleSplit
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

Creating a function to calculate best model for this problem

```
def find_best_model(X, y):
    models = {
        'logistic_regression': {
            'model': LogisticRegression(solver='lbfgs', multi_class='auto'),
            'parameters': {
                'C': [1,5,10]
              }
        },

        'decision_tree': {
            'model': DecisionTreeClassifier(splitter='best'),
            'parameters': {
                'criterion': ['gini', 'entropy'],
                'max_depth': [5,10]
            }
        },

        'random_forest': {
            'model': RandomForestClassifier(criterion='gini'),
            'parameters': {
                'n_estimators': [10,15,20,50,100,200]
            }
        },

        'svm': {
            'model': SVC(gamma='auto'),
            'parameters': {
                'C': [1,10,20],
                'kernel': ['rbf','linear']
            }
        }
```

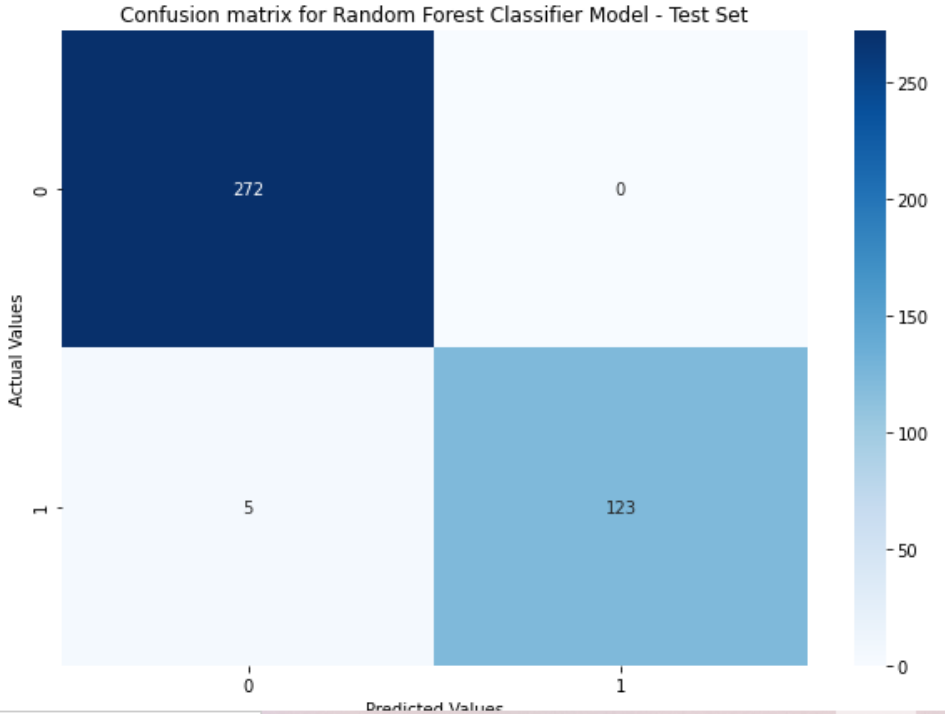
```
    }

    scores = []
    cv_shuffle = ShuffleSplit(n_splits=5, test_size=0.20, random_state=0)

    for model_name, model_params in models.items():
        gs = GridSearchCV(model_params['model'], model_params['parameters'], cv = cv_shuffle, return_train_score=False)
        gs.fit(X, y)
        scores.append({
            'model': model_name,
            'best_parameters': gs.best_params_,
            'score': gs.best_score_
        })

    return pd.DataFrame(scores, columns=['model','best_parameters','score'])

find_best_model(X_train, y_train)
```

ut[23]:

	model	best_parameters	score
0	logistic_regression	{'C': 10}	0.763125
1	decision_tree	{'criterion': 'gini', 'max_depth': 10}	0.901875
2	random_forest	{'n_estimators': 15}	0.951250
3	svm	{'C': 20, 'kernel': 'rbf'}	0.869375

Plotting the confusion matrix

```
plt.figure(figsize=(10,7))
p = sns.heatmap(cm, annot=True, cmap="Blues", fmt='g')
plt.title('Confusion matrix for Random Forest Classifier Model - Test Set')
plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
plt.show()
```



Accuracy Score

```
In [31]: # Accuracy Score
         score = round(accuracy_score(y_test, y_pred),4)*100
         print("Accuracy on test set: {}%".format(score))

         Accuracy on test set: 98.75%
```

Classification Report

In [32]:
```
# Classification Report
print(classification_report(y_test, y_pred))
```

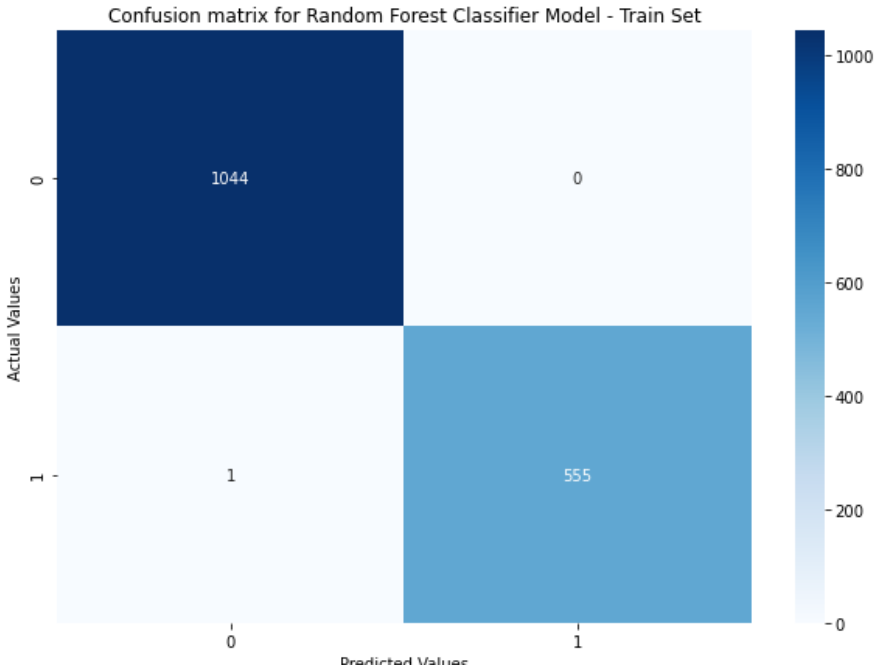
```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       272
           1       1.00      0.96      0.98       128

    accuracy                           0.99       400
   macro avg       0.99      0.98      0.99       400
weighted avg       0.99      0.99      0.99       400
```

In [33]:
```
# Creating a confusion matrix for training set
y_train_pred = classifier.predict(X_train)
cm = confusion_matrix(y_train, y_train_pred)
cm
```

Out[33]:
```
array([[1044,    0],
       [   1,  555]], dtype=int64)
```

Plotting the confusion matrix for train set

```
plt.figure(figsize=(10,7))
p = sns.heatmap(cm, annot=True, cmap="Blues", fmt='g')
plt.title('Confusion matrix for Random Forest Classifier Model - Train Set')
plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
plt.show()
```



Accuracy Score

```
: # Accuracy Score
score = round(accuracy_score(y_train, y_train_pred),4)*100
print("Accuracy on trainning set: {}%".format(score))
```

```
Accuracy on trainning set: 99.94%
```

Classification Report

```
In [36]: # Classification Report
print(classification_report(y_train, y_train_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1044
           1       1.00      1.00      1.00       556

    accuracy                           1.00      1600
   macro avg       1.00      1.00      1.00      1600
weighted avg       1.00      1.00      1.00      1600
```

Creating a function for prediction

```
def predict_diabetes(Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DPF, Age):
    preg = int(Pregnancies)
    glucose = float(Glucose)
    bp = float(BloodPressure)
    st = float(SkinThickness)
    insulin = float(Insulin)
    bmi = float(BMI)
    dpf = float(DPF)
    age = int(Age)

    x = [[preg, glucose, bp, st, insulin, bmi, dpf, age]]
    x = sc.transform(x)

    return classifier.predict(x)
```

Predictions:

In [38]:
```
# Prediction 1
# Input sequence: Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DPF, Age
prediction = predict_diabetes(2, 81, 72, 15, 76, 30.1, 0.547, 25)[0]
if prediction:
  print('Oops! You have diabetes.')
else:
  print("Great! You don't have diabetes.")
```

```
Great! You don't have diabetes.
```

In [39]:
```
# Prediction 2
# Input sequence: Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DPF, Age
prediction = predict_diabetes(1, 117, 88, 24, 145, 34.5, 0.403, 40)[0]
if prediction:
  print('Oops! You have diabetes.')
else:
  print("Great! You don't have diabetes.")
```

```
Oops! You have diabetes.
```

In [40]:
```
# Prediction 3
# Input sequence: Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DPF, Age
prediction = predict_diabetes(5, 120, 92, 10, 81, 26.1, 0.551, 67)[0]
if prediction:
  print('Oops! You have diabetes.')
else:
  print("Great! You don't have diabetes.")
```

```
Great! You don't have diabetes.
```

Conclusion:

After using all these patient records, we are able to build a machine learning model (random forest – best one) to accurately predict whether or not the patients in the dataset have diabetes or not along with that we were able to draw some insights from the data via data analysis and visualization.