ABSTRACT

The healthcare sector has swiftly grown to be particularly interested in the idea of machine learning. Medical data sets used in research predictions and analyses aid with appropriate treatment and precautions in illness prevention. the kinds of algorithms that can aid in decision-making and prediction in machine learning. We also talk about how machine learning is being used in the medical industry, with a particular emphasis on predicting diabetes.

One of the diseases that is spreading fastest in the world is diabetes, which needs to be monitored constantly. We investigate various machine learning techniques that will aid in the early diagnosis of this disease to verify this. This work explores several facets of machine learning and the different kinds of algorithms that can aid in prediction and decision-making. The scientific community's forecasts and analyses of medical datasets help the public take the necessary care and safeguards to stay healthy.

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.

INTRODUCTION

According to a 2019 World Health Organization data, there were 463 million cases of diabetes worldwide. There were also 1.5 million fatalities, therefore it is easy to infer that diabetes is a dangerous and chronic condition in a significant number of cases.

Chronic, lifelong diabetes mellitus is brought on by abnormally high blood sugar levels. In the medical industry, classification algorithms are frequently employed to divide data into various classes based on restrictions placed on a single classifier. Since diabetes interferes with the body's ability to produce the hormone insulin, improper carbohydrate metabolism and elevated blood sugar levels occur.

Many researchers conduct experiments to diagnose diseases using different machine learning approach classification algorithms such as logistic regression, Decision Tree KNN, SVM, Random Forest classifier because researchers have proven demonstrated that machine learning algorithms are more effective.

For our project, we use the random forest algorithm for the prediction. It is based on the idea of ensemble learning, which is a method of combining various classifiers to address complex issues and enhance model performance. Random Forest is a classifier that contains several decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset.

Random Forest can perform both Classification and Regression tasks. It is capable of handling large datasets with high dimensionality. It enhances the accuracy of the model and prevents the overfitting issue.

The database used by us is Pima Indians Diabetes Dataset. The datasets consist of several medical predictor variables and one target variable, Outcome.

- Pregnancies: Number of times pregnant
- Glucose: Plasma glucose concentration 2 hours in an oral glucose tolerance test
- Blood Pressure: Diastolic blood pressure (mm Hg)
- Skin Thickness: Triceps skin fold thickness (mm)
- Insulin: 2-Hour serum insulin (mu U/ml)
- BMI: Body mass index (weight in kg/(height in m)^2)
- Diabetes Pedigree Function: Diabetes pedigree function
- Age: Age (years)
- Outcome: Class variable (0 or 1)

The objective of this project is:

1. Analyse the dataset under the point of view of a Dietitian.
2. Apply machine learning techniques resulting in bridging the gap between datasets and human knowledge.

OUR KEY TAKEAWAYS:

- Data analysis: Here one will get to know about how the data analysis part is done in a data science life cycle.

- Exploratory data analysis: EDA is one of the most important step in data science project life cycle and here one will need to know that how to make inferences from the visualizations and data analysis

- Model building: Here we will be using 4 ML models and then we will choose the best performing model.

- Saving model: Saving the best model using pickle to make the prediction from real data.

MODEL BUILDING:

This is most important phase which includes model building for prediction of diabetes. In this we have implemented various machine learning algorithms which are discussed for diabetes prediction.

Procedure of Proposed Methodology-

Step 1: Import required libraries, Import diabetes dataset.

Step 2: Pre-process data to remove missing data.

Step 3: Perform percentage split of 80% to divide dataset as Training set and 20% to Test set.

Step 4: Select the machine learning algorithm i.e., K- Nearest Neighbour, Support Vector Machine, Decision Tree, Logistic regression, Random Forest, and Gradient boosting algorithm.

Step 5: Build the classifier model for the mentioned ma- chine learning algorithm based on training set.

Step 6: Test the Classifier model for the mentioned ma- chine learning algorithm based on test set.

Step 7: Perform Comparison Evaluation of the experimental performance results obtained for each classifier

Step 8: After analysing based on various measures conclude the best performing algorithm.

IMPLEMENTATION

Diabetes prediction using machine learning (4 algorithms)

In this article we will be predicting that whether the patient have diabetes or not basis on the features we will provide to our machine learning model and for that we will be using famous Pima Indians Diabetes Database - https://www.kaggle.com/uciml/pima-indians-diabetes-database

Importing Libraries

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

sns.set()

from mlxtend.plotting import plot_decision_regions
import missingno as msno
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import classification_report
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

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

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

[3]:
```
diabetes_df.head()
```

[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

Exploratory Data Analysis (EDA)

In [4]:
```
# Now let' see that what are the columns available in our dataset.
diabetes_df.columns
```

Out[4]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [5]:
```
# Information about the dataset
diabetes_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   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    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [6]:
```
# To know more about the dataset
diabetes_df.describe()
```

Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

In [7]:
```
# To know more about the dataset with transpose - here T is for the transpose
diabetes_df.describe().T
```

Out[7]:

	count	mean	std	min	25%	50%	75%	max
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199.00
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000	122.00
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000	99.00
Insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000	846.00
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000	67.10
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2.42
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	81.00
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.00

In [57]:
```
# Now let's check that if our dataset have null values or not
diabetes_df.isnull().head(10)
```

Out[57]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False
5	False	False	False	False	False	False	False	False	False
6	False	False	False	False	False	False	False	False	False
7	False	False	False	False	False	False	False	False	False
8	False	False	False	False	False	False	False	False	False
9	False	False	False	False	False	False	False	False	False

In [9]:
```
# Now let's check that if our dataset have null values or not
diabetes_df.isnull().sum()
```

Out[9]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Here from above code we first checked that is there any null values from isnull() function then we are going to take the sum of all those missing values from 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 value which is not good for the authenticity of the dataset. Hence we will first replace the 0 value to NAN value then start the imputation process.

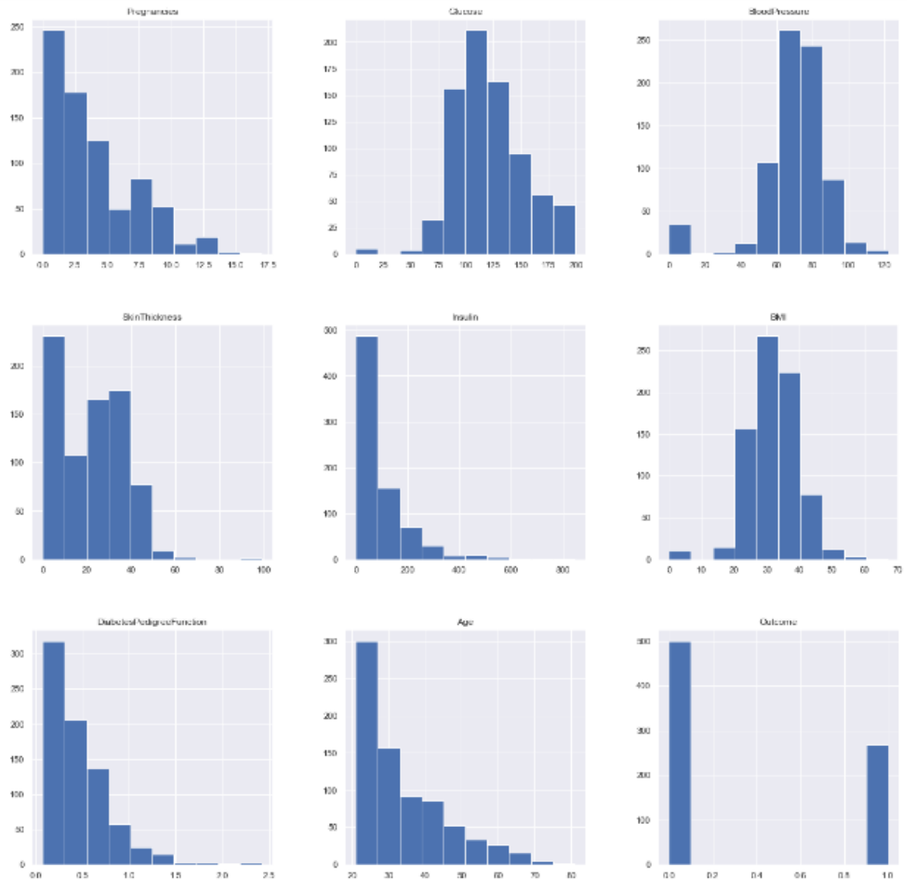
In [10]:
```
diabetes_df_copy = diabetes_df.copy(deep = True)
diabetes_df_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = diabetes_df_copy[['Glucose','BloodPressure','Skin

# Showing the Count of NANs
print(diabetes_df_copy.isnull().sum())
```

```
Pregnancies                 0
Glucose                     5
BloodPressure              35
SkinThickness             227
Insulin                   374
BMI                        11
DiabetesPedigreeFunction    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 better Imputation approach i.e to apply mean values of each column to the null values of the respectove columns.

Data Visualization

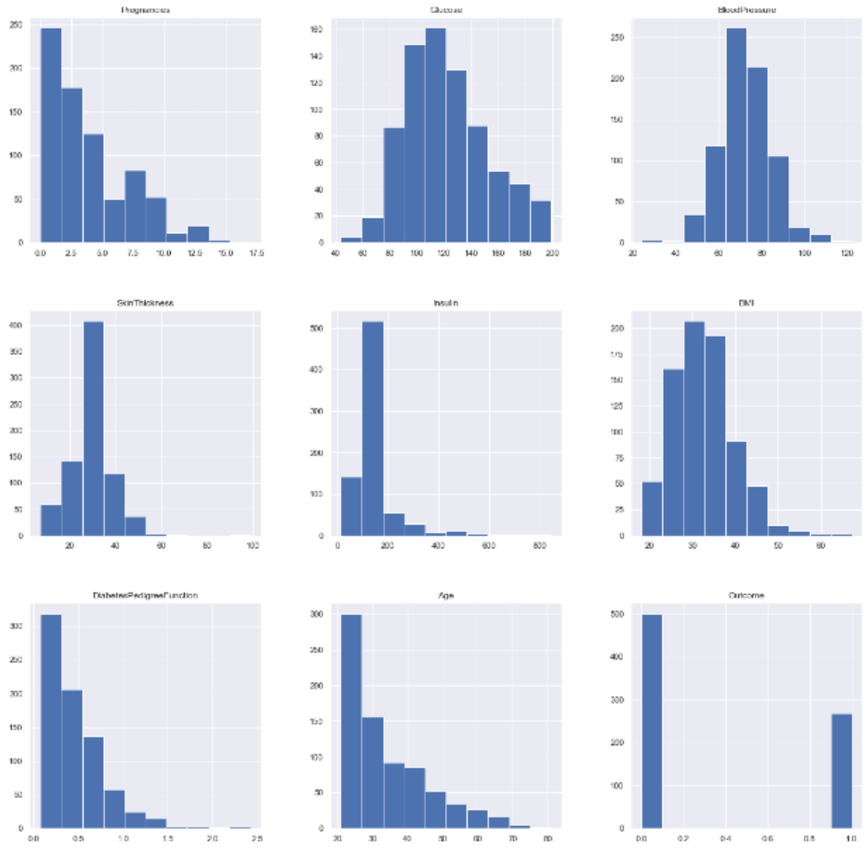
In [11]:
```
# Plotting the data distribution plots before removing null values
p = diabetes_df.hist(figsize = (20,20))
```



So here we have seen the distribution of each features whether it is dependent data aur independent data and one thing which could always strikes that why do we need to see the distribution of data ? So the ans is simple it is the best way to start the analysis of the dataset as it shows the occurence of evry kind of value in the graphical structure which in turn let us know the range of the data

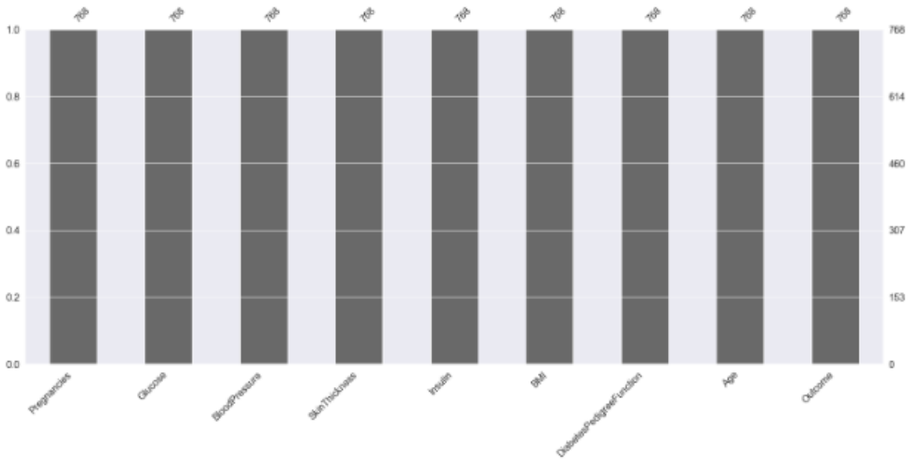
In [12]:
```
# Now we will be imputing the mean value of the column to each missing value of that particular column
diabetes_df_copy['Glucose'].fillna(diabetes_df_copy['Glucose'].mean(), inplace = True)
diabetes_df_copy['BloodPressure'].fillna(diabetes_df_copy['BloodPressure'].mean(), inplace = True)
diabetes_df_copy['SkinThickness'].fillna(diabetes_df_copy['SkinThickness'].median(), inplace = True)
diabetes_df_copy['Insulin'].fillna(diabetes_df_copy['Insulin'].median(), inplace = True)
diabetes_df_copy['BMI'].fillna(diabetes_df_copy['BMI'].median(), inplace = True)
```

In [13]:
```
# Plotting the distributions after removing the NAN values
p = diabetes_df_copy.hist(figsize = (20,20))
```



Here we are again using the hist plot to see the distribution of the dataset but this time we are using this visualization to see the changes that we can see after those null values are removed from the dataset and we can clearly see the difference for example - In age column after removal of the null values we can see that there is a spike at the range of 50 to 100 which is quite logical as well.

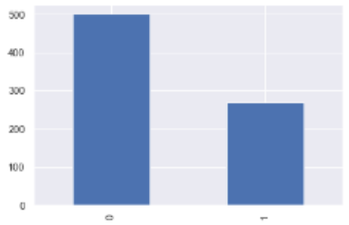
In [14]:
```
# Plotting Null Count Analysis Plot
p = msno.bar(diabetes_df)
```



Inference : Now in the above graph also we can clearly see that there are no null values in the dataset

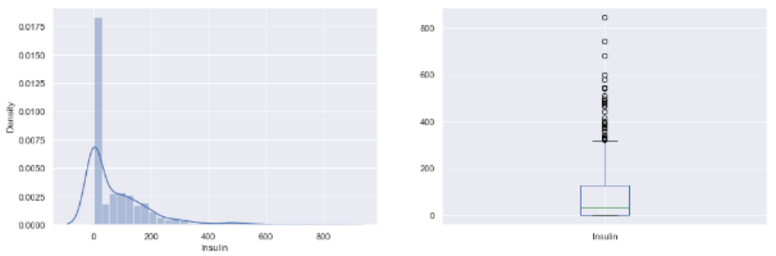
In [15]:
```
# Now, let's check that how well our outcome column is balanced
color_wheel = {1: "#0392cf", 2: "#7bc043"}
colors = diabetes_df["Outcome"].map(lambda x: color_wheel.get(x + 1))
print(diabetes_df.Outcome.value_counts())
p=diabetes_df.Outcome.value_counts().plot(kind="bar")
```

```
0    500
1    268
Name: Outcome, dtype: int64
```



Inference : Here from the above visualization it is clearly visible that our dataset is completely imbalanced infact the number of patient who is diabetic is half of the patients who are non-diabetic

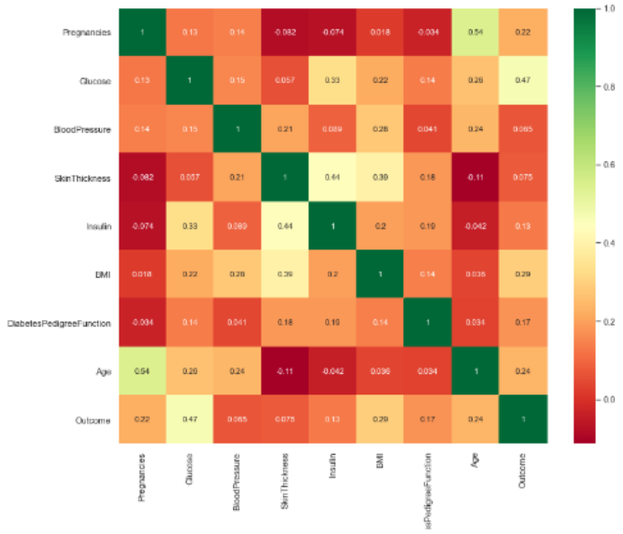
In [58]:
```
plt.subplot(121), sns.distplot(diabetes_df['Insulin'])
plt.subplot(122), diabetes_df['Insulin'].plot.box(figsize=(16,5))
plt.show()
```



That's how distplot can be helpful where one will able to see the distribution of the data as well as with the help of boxplot one can see the outliers in that column and other information too which can be derieved by the box and whiskers plot

Correlation between all the features

In [16]:
```
# Correlation between all the features before cleaning
plt.figure(figsize=(12,10))
p = sns.heatmap(diabetes_df.corr(), annot=True,cmap ='RdYlGn')  # seaborn has an easy method to showcase heatmap
```



Scaling the data

In [18]:
```
diabetes_df_copy.head()
```

Out[18]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35.0	125.0	33.6	0.627	50	1
1	1	85.0	66.0	29.0	125.0	26.6	0.351	31	0
2	8	183.0	64.0	29.0	125.0	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

In [19]:
```
sc_X = StandardScaler()
X =  pd.DataFrame(sc_X.fit_transform(diabetes_df_copy.drop(["Outcome"],axis = 1),), columns=['Pregnancies', 'Glucose', 'BloodPr
X.head()
```

Out[19]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	0.639947	0.865108	-0.033518	0.670643	-0.181541	0.166619	0.468492	1.425995
1	-0.844885	-1.206162	-0.529859	-0.012301	-0.181541	-0.852200	-0.365061	-0.190672
2	1.233880	2.015813	-0.695306	-0.012301	-0.181541	-1.332500	0.604397	-0.105584
3	-0.844885	-1.074652	-0.529859	-0.695245	-0.540642	-0.633881	-0.920763	-1.041549
4	-1.141852	0.503458	-2.680669	0.670643	0.316566	1.549303	5.484909	-0.020496

That's how our dataset will be looking like when it is scaled down or we can see every value now is on the same scale which will help our ML model to give better result

In [20]:
```
y = diabetes_df_copy.Outcome
```

In [21]:
```
y
```

Out[21]:
```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

Model Building

Splitting the dataset

In [24]:
```
#Splitting the dataset

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

In [25]:
```
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.33,
                                                    random_state=7)
```

In [29]:
```
#Check columns with zero values - checking this time so that right data should go for model training

print("Total number of rows: {0}", format(len(diabetes_df)))
print("Number of rows missing Pregnancies: {0}",
      format(len(diabetes_df.loc[diabetes_df['Pregnancies']==0])))
print("Number of rows missing Glucose: {0}"
      , format(len(diabetes_df.loc[diabetes_df['Glucose']==0])))
print("Number of rows missing BloodPressure: {0}",
      format(len(diabetes_df.loc[diabetes_df['BloodPressure']==0])))
print("Number of rows missing SkinThickness: {0}",
      format(len(diabetes_df.loc[diabetes_df['SkinThickness']==0])))
print("Number of rows missing Insulin: {0}",
      format(len(diabetes_df.loc[diabetes_df['Insulin']==0])))
print("Number of rows missing BMI: {0}",
      format(len(diabetes_df.loc[diabetes_df['BMI']==0])))
print("Number of rows missing DiabetesPedigreeFunction: {0}",
      format(len(diabetes_df.loc[diabetes_df['DiabetesPedigreeFunction']==0])))
print("Number of rows missing Age: {0}", format(len(diabetes_df.loc[diabetes_df['Age']==0])))
```

```
Total number of rows: {0} 768
Number of rows missing Pregnancies: {0} 111
Number of rows missing Glucose: {0} 5
Number of rows missing BloodPressure: {0} 35
Number of rows missing SkinThickness: {0} 227
Number of rows missing Insulin: {0} 374
Number of rows missing BMI: {0} 11
Number of rows missing DiabetesPedigreeFunction: {0} 0
Number of rows missing Age: {0} 0
```

In [30]:
```
#Imputing zeros values in the dataset

from sklearn.impute import SimpleImputer
import numpy as np

fill_values = SimpleImputer(missing_values=0, strategy='mean')
X_train = fill_values.fit_transform(X_train)
X_test = fill_values.fit_transform(X_test)
```

In [31]:
```
#Building the model using RandomForest

from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators=200)
rfc.fit(X_train, y_train)
```

Out[31]:
```
RandomForestClassifier(n_estimators=200)
```

In [32]:
```
# On training data
rfc_train = rfc.predict(X_train)
from sklearn import metrics

print("Accuracy_Score =", format(metrics.accuracy_score(y_train, rfc_train)))
```

```
Accuracy_Score = 1.0
```

In [33]:
```
predictions = rfc.predict(X_test)
```

In [34]:
```
#Getting the accuracy score for Random Forest

from sklearn import metrics

print("Accuracy_Score =", format(metrics.accuracy_score(y_test, predictions)))
```

```
Accuracy_Score = 0.7677165354330708
```

In [35]:
```
from sklearn.metrics import classification_report, confusion_matrix

print(confusion_matrix(y_test, predictions))
print(classification_report(y_test,predictions))
```

```
[[133  29]
 [ 30  62]]
              precision    recall  f1-score   support

           0       0.82      0.82      0.82       162
           1       0.68      0.67      0.68        92

    accuracy                           0.77       254
   macro avg       0.75      0.75      0.75       254
weighted avg       0.77      0.77      0.77       254
```

In [36]:
```
#Building the model using DecisionTree

from sklearn.tree import DecisionTreeClassifier

dtree = DecisionTreeClassifier()
dtree.fit(X_train, y_train)
```

Out[36]:
```
DecisionTreeClassifier()
```

In [37]:
```
predictions = dtree.predict(X_test)
```

In [38]:
```
#Getting the accuracy score for Decision Tree

from sklearn import metrics

print("Accuracy Score =", format(metrics.accuracy_score(y_test,predictions)))
```

```
Accuracy Score = 0.7322834645669292
```

In [39]:
```
from sklearn.metrics import classification_report, confusion_matrix

print(confusion_matrix(y_test, predictions))
print(classification_report(y_test,predictions))
```

```
[[126  36]
 [ 32  60]]
              precision    recall  f1-score   support

           0       0.80      0.78      0.79       162
           1       0.62      0.65      0.64        92

    accuracy                           0.73       254
   macro avg       0.71      0.71      0.71       254
weighted avg       0.73      0.73      0.73       254
```

In [40]:
```
#Building model using XGBoost

from xgboost import XGBClassifier

xgb_model = XGBClassifier(gamma=0)
xgb_model.fit(X_train, y_train)
```

```
[01:21:35] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.
```

Out[40]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=4, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [41]:
```
xgb_pred = xgb_model.predict(X_test)
```

In [42]:
```
#Getting accuracy score for XGBoost

from sklearn import metrics

print("Accuracy Score =", format(metrics.accuracy_score(y_test, xgb_pred)))
```

```
Accuracy Score = 0.7401574803149606
```

In [43]:
```
#Metrics for XGBoost
from sklearn.metrics import classification_report, confusion_matrix

print(confusion_matrix(y_test, xgb_pred))
print(classification_report(y_test,xgb_pred))
```

```
[[127  35]
 [ 31  61]]
              precision    recall  f1-score   support

           0       0.80      0.78      0.79       162
           1       0.64      0.66      0.65        92

    accuracy                           0.74       254
   macro avg       0.72      0.72      0.72       254
weighted avg       0.74      0.74      0.74       254
```

In [44]:
```
#Building the model using Support Vector Machine (SVM)

from sklearn.svm import SVC

svc_model = SVC()
svc_model.fit(X_train, y_train)
```

Out[44]:
```
SVC()
```

In [45]:
```
#Predict
svc_pred = svc_model.predict(X_test)
```

In [46]:
```
#Accuracy score for SVM
from sklearn import metrics

print("Accuracy Score =", format(metrics.accuracy_score(y_test, svc_pred)))
```

```
Accuracy Score = 0.7401574803149606
```

In [47]:
```
#Metrics for SVM
from sklearn.metrics import classification_report, confusion_matrix

print(confusion_matrix(y_test, svc_pred))
print(classification_report(y_test,svc_pred))
```

```
[[143  19]
 [ 47  45]]
              precision    recall  f1-score   support

           0       0.75      0.88      0.81       162
           1       0.70      0.49      0.58        92

    accuracy                           0.74       254
   macro avg       0.73      0.69      0.69       254
weighted avg       0.73      0.74      0.73       254
```

Conclusion from model building

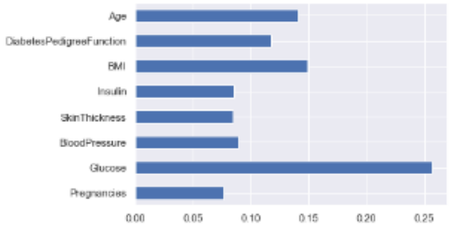
Therefore Random forest is the best model for this prediction since it has an accuracy_score of 0.78

In [48]:
```
#Getting feature importances
rfc.feature_importances_
```

Out[48]:
```
array([0.07684946, 0.25643635, 0.08952599, 0.08437176, 0.08552636,
       0.14911634, 0.11751284, 0.1406609 ])
```

In [49]:
```
#Plotting feature importances
(pd.Series(rfc.feature_importances_, index=X.columns)
   .plot(kind='barh'))
```

Out[49]:
```
<AxesSubplot:>
```



Here from the above graph it is clearly visible that Glucose as a feature has the most importance in this dataset.

In [50]:
```
#Printing prediction probabilities for the test data
print('Prediction Probabilities')
rfc.predict_proba(X_test)
```

```
Prediction Probabilities
```

Out[50]:
```
array([[0.955, 0.045],
       [0.165, 0.835],
       [0.52 , 0.48 ],
       [0.86 , 0.14 ],
       [0.42 , 0.58 ],
       [0.41 , 0.59 ],
       [0.91 , 0.09 ],
       [0.84 , 0.16 ],
       [0.145, 0.855],
       [0.79 , 0.21 ],
       [0.08 , 0.92 ],
       [0.905, 0.095],
       [0.255, 0.745],
       [0.075, 0.925],
       [0.7  , 0.3  ],
       [0.77 , 0.23 ],
       [0.81 , 0.19 ],
       [0.51 , 0.49 ],
```

Saving model

In [51]:
```
import pickle

# Firstly we will be using the dump() function to save the model using pickle
saved_model = pickle.dumps(rfc)

# Then we will be loading that saved model
rfc_from_pickle = pickle.loads(saved_model)

# Lastly, after loading that model we will use this to make predictions
rfc_from_pickle.predict(X_test)
```

Out[51]:
```
array([0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0,
       1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
       1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0,
       1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1,
       1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1], dtype=int64)
```

In [53]:
```
diabetes_df.head()
```

Out[53]:

	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

In [54]:
```
diabetes_df.tail()
```

Out[54]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

In [55]:
```
# putting datapoints in the model it will either return 0 or 1 i.e. person suffering from diabetes or not
rfc.predict([[0,137,40,35,168,43.1,2.228,33]]) #4th patient
```

Out[55]:
```
array([1], dtype=int64)
```

In [56]:
```
# putting datapoints in the model it will either return 0 or 1 i.e. person suffering from diabetes or not
rfc.predict([[10,101,76,48,180,32.9,0.171,63]])  # 763 th patient
```

Out[56]:
```
array([0], dtype=int64)
```

Conclusion

After using all these these patient records, we are able to build a machine learning model to accurately predict whether or not the patients in the dataset have diabetes or not.

LIBRARIES USED:

Pandas: Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real-world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis/manipulation tool available in any language. It is already well on its way toward this goal.

NumPy: NumPy is a Python library used for working with arrays.It also has functions for working in the domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely. NumPy stands for Numerical Python.
In Python we have lists that serve the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in NumPy is called ND array, it provides a
lot of supporting functions that make working with ND array very easy. Arrays are
very frequently used in data science, where speed and resources are very important. NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently. This behaviour
is called locality of reference in computer science. This is the main reason why NumPy is faster than lists. Also it is optimized to work with the latest CPU architectures. NumPy is a Python library and is written partially in Python, but most of the
parts that require fast computation are written in C or C++.

Matplotlib: Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in 2002.
One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

Seaborn: Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

Sklearn: Scikit-learn is probably the most useful library for machine learning in Python. The sklearn library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction.

Statsmodels: statsmodels is a Python module that provides classes and functions for the estimation of many different statistical models, as well as for conducting statistical tests, and statistical data exploration. An extensive list of result statistics are available for each estimator. The results are tested against existing statistical packages to ensure that they are correct.

REQUIREMENTS:

Hardware requirements:

Processor : Any Update Processor
Ram : Min 4GB
Hard Disk : Min 100GB

Software requirements:

Operating System : Windows family
Technology : Python3.7
IDE : Jupyter notebook

CONCLUSION

This project is a small contribution to the present existing methods of diabetes detection by proposing a system that can be used as an assistive tool in identifying the patients at greater risk of being diabetic. This project achieves this by analysing many key factors like the patient's blood glucose level, body mass index, etc., using various machine learning models and through retrospective analysis of patients' medical records.

The project predicts the onset of diabetes in a person based on the relevant medical details collected. When the person enters all the relevant medical data required, this data is then passed on to the trained model for it to make predictions whether the person is diabetic or non-diabetic the model then makes the prediction with an accuracy which is good and reliable.

We used 70% of data for training and 30% of data for testing. In this ratio of data splitting Here we found that Random Forest Classifier predicted with a higher accuracy than the other methods of prediction.

REFERENCES:

- https://www.kaggle.com/
- https://www.simplilearn.com/
- https://www.analyticsvidhya.com
- https://www.javatpoint.com/
- https://www.tutorialspoint.com/