TOPIC:

DIABETES PREDICTION USING DATA MINING

TEAM MEMBERS:

HARSHIT RASTOGI -20BDS0138

RAHUL KUMAR SAHANI – 20BDS0126

AMEY MAHABALESHWAR BHAT- 20BDS0133

PRACHI BALODIA-20BDS0177

ABSTRACT :

Diabetes is one of deadliest diseases in the world. World Health Organization reports says that around 422 million people have diabetes worldwide.. As per the existing system in Sri Lanka, patients have to visit a diagnostic center, consult their doctor and wait for a day or more to get their result. Moreover, every time they want to get their diagnosis report, they have to waste their money in vain. But with the rise of Machine Learning approaches, we have been able to find a solution to this problem using data mining. **Data mining** is one of the key areas of Machine learning. It plays a significant role in diabetes research because It has the ability to extract hidden knowledge from a huge amount of diabetes related data. The aim of this project is to develop a system which can predict whether the patient has diabetes or not. Furthermore, predicting the disease early leads to treatment of the patients before it becomes critical. Now our project has focused on developing a system based on three classification methods namely, **Decision Tree, XG Boost, Random Forests and Support Vector Machine algorithms**. At first we will collect the dataset of various persons and will be gathering those information in a single file. In this project we will be using **jupyter notebook along with python 3** to perform the code. **There are 7 steps that we follow in this project:**

1. Data Collection

2. Data Preparation

3. Choosing a model

4. Training the model

5. Evaluating the model

6. Parameter tuning

7. Making prediction

After making the prediction we will compare the accuracy rate of these three algorithms and the algorithm which has the best accuracy rate will be used in predicting whether the person has diabetes or not. Therefore this project concentrated on providing different prediction methods of diabetes.

Objective:

The objective is to predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset.

0– Absence of Diabetes

1 – Presence of Diabetes

About the dataset:

There are 8 variables that we used in the data set these are:

pregnancies – Number of times pregnant

Glucose – Plasma glucose concentration a 2 hours in an oral glucose tolerance test

Blood Pressure – Diastolic blood pressure (mm Hg)

Skin Thickness – Triceps skinfold 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) 268 of 768 are 1, the others are 0

PROPOSED METHODOLOGY:

Below mentioned are the steps involved in the completion of this project:

1: Collect dataset containing details of patients from the open source platforms

2: Write a code to extract the required features from the data

3: Divide the data set into training and testing sets.

4: Run selected machine learning and deep neural network algorithms like XGBoost on the data set.

5: Write a code for displaying the evaluation result considering accuracy metrics.

Compare the obtained results for trained models and specify which it is better.

Literature Review:

Diabetes is one of the fastest growing chronic disease in the world which has affected millions of people around the globe. Many people don't go to doctor for health check -up due to which they remain unknown about the disease and continue to eat sweets and sweet related items which affects their heart, lungs ,eyes and several part of the brain. Diabetes has become one of the major public health problems in today's world due to its prevalence in children and adult population and the associated high cost of management and healthcare. So now in this project we will mainly classify the commonly used data mining methods for diabetes diagnosis and prediction based on the underlying model used . Then we will compare them based on their key parameters and metrics and then we will finalise the model that has the highest prediction among all models and that will be useful for predicting the probabilities of which likely to have a diabetes. The data set is collected from various open source platforms and also conducting a survey by keeping in mind all the factors which are helpful in predicting the diabetes like glucose level, blood pressure, Body Mass Index and so on and it is put in an excel sheet so that we can retrieve it on jupyter notebook and work on it . Here we have then explore and visualise the data by considering the all the factors that we have surveyed. After that we have to build and train the model with the help of various classification based techniques . Classification is a supervised learning process in which a class of objects is classified in order to predict any classes of future objects. The four data mining methods that we will be using mainly for this project are Random Forests, Decision Tree XG Boost and Support Vector machine algorithm.

Decision trees are widely used models for classification and regression tasks. Essentially, they learn a hierarchy of if/else questions, leading to a decision. Learning a decision tree means learning the sequence of if/else questions that gets us to the true answer most quickly.

In the data mining , these questions are called tests (not to be confused with the test set, which is the data we use to test to see how generalizable our model is). To build a tree, the algorithm searches over all possible tests and finds the one that is most informative about the target variable. This algorithm is easy to read and interpret. One of the advantages of decision trees is that their outputs are easy to read and interpret without requiring statistical knowledge, Easy to prepare. Less data cleaning required.

XGBoost is one of the most popular algorithms these days. XGBoost stands for eXtreme Gradient Boosting. Regardless of the type of prediction task at hand; regression or classification. XGBoost is an implementation of gradient boosted

decision trees designed for speed and performance. XGBoost is a scalable and accurate implementation of gradient boosting machines and it has proven to push the limits of computing power for boosted trees algorithms as it was built and developed for the sole purpose of model performance and computational speed. Moreover XG Boost algorithm is also flexible .

In machine learning, support-vector machines (SVMs, also support-vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. SVM works relatively well when there is a clear margin of separation between classes. SVM is more effective in high dimensional spaces. SVM is effective in cases where the number of dimensions is greater than the number of samples. SVM is relatively memory efficient.

Random forest is a supervised learning algorithm. ... The general idea of the bagging method is that a combination of learning models increases the overall result. Put simply: random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction. The random forest technique can also handle big data with numerous variables running into thousands. It can automatically balance data sets when a class is more infrequent than other classes in the data.

Now after building the model we will check the accuracy rate of all the all the four algorithms and the algorithm which have the best accuracy rate will be used to determine the prediction probability to check whether the person can have diabetes or not.

REFERNCES :

1.Agarwal, A., & Saxena, A. (2019, March). Analysis of machine learning algorithms and obtaining highest accuracy for prediction of diabetes in women. In 2019 6th International Conference on Computing for Sustainable Global Development (INDIACom) (pp. 686-690). IEEE.

2.Shetty, D., Rit, K., Shaikh, S., & Patil, N. (2017, March). Diabetes disease prediction using data mining. In 2017 international conference on innovations in information, embedded and communication systems (ICIIECS) (pp. 1-5). IEEE.

3. Repalli, P. (2011). Prediction on diabetes using data mining approach. Oklahoma State University.

4.Woldemichael, F. G., & Menaria, S. (2018, May). Prediction of diabetes using data mining techniques. In 2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI) (pp. 414-418). IEEE.

5. (•‿•): Islam, M. F., Ferdousi, R., Rahman, S., & Bushra, H. Y. (2020). Likelihood prediction of diabetes at early stage using data mining techniques. In Computer Vision and Machine Intelligence in Medical Image Analysis (pp. 113-125). Springer, Singapore.

6.NandanaPathiragehttps://www.researchgate.net/publication/338581650_Diabetic_Prediction_System_Using_Data_Mining

IMPLEMENTATION :

So let us begin by importing the required libraries. We will import data analysis libraries (pandas, numpy) and visualization libraries (matplotlib, seaborn) . In addition to that, we will also import warnings so that warnings are hidden from the notebook in case there is.

it is important to set %matplotlib inline+ to show the visualizations on the notebook.

```
#Import required libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

#Ignore warnings
warnings.filterwarnings('ignore')

#Seaborn visualization set up
%matplotlib inline
sns.set_style('darkgrid')
```

Import the dataset and do a few checks as follows;

```
#Reading the dataset

data = pd.read_csv('diabetes.csv')
data.head(10)
```

	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
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1

As you can note, the dataset has 9 columns.

```
data.shape
```

```
(768, 9)
```

Let us now check the columns and their data types.

```
data.info()
```

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

From the dataset, 7 columns have int data type and 2 columns have float data type.

```
data.columns
```

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

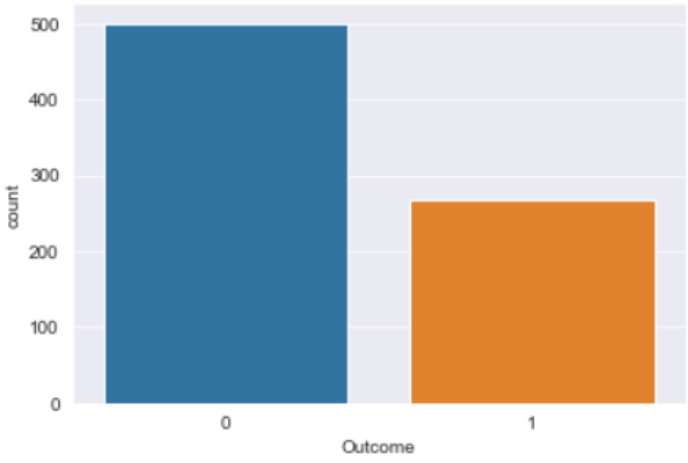
Data Exploration and Visualisation:

In this section, we will create graphs to displays different distributions of the data and available relationships to allow us to understand it much better. This is a very critical section since it determines how the model will be built.

- Checking the distribution of the target variable

```
#Plotting the distribution of Outcome
sns.countplot(x= 'Outcome', data = data)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x220dc6e8160>
```



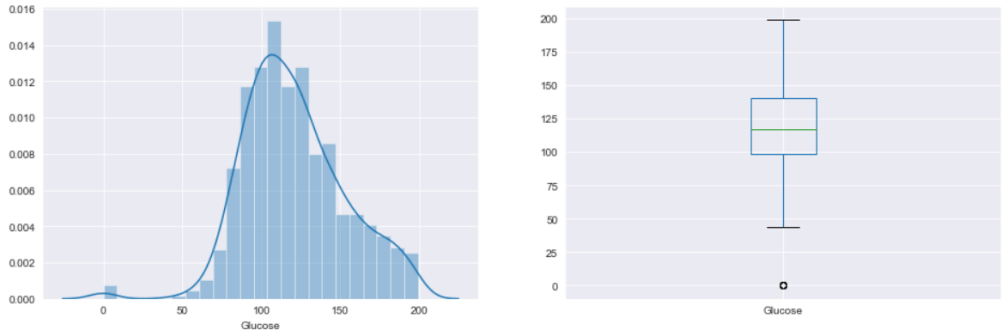
From the plot above, the data contains more cases without diabetes (0) than those with diabetes (1).

- Checking the distribution of the predictor variables

Here, we will use both distplot and boxplot as shown below. Let us plot each variable to show its distribution in the dataset.

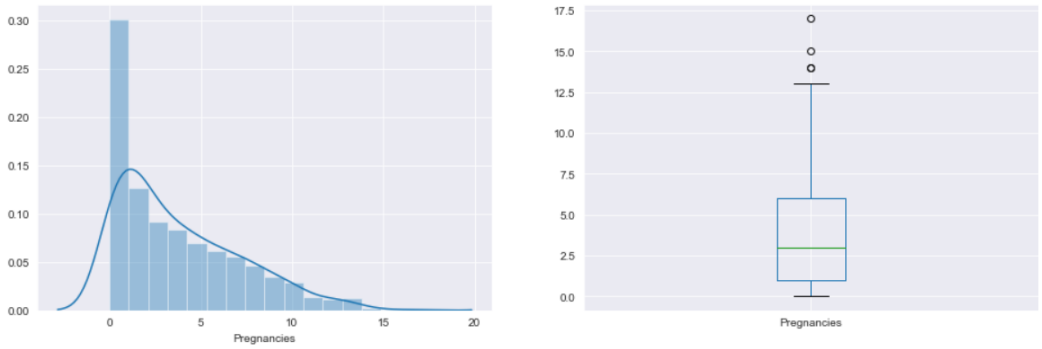
Distribution of Glucose

```
plt.figure(1)
plt.subplot(121), sns.distplot(data['Glucose'])
plt.subplot(122), data['Glucose'].plot.box(figsize=(16,5))
plt.show()
```



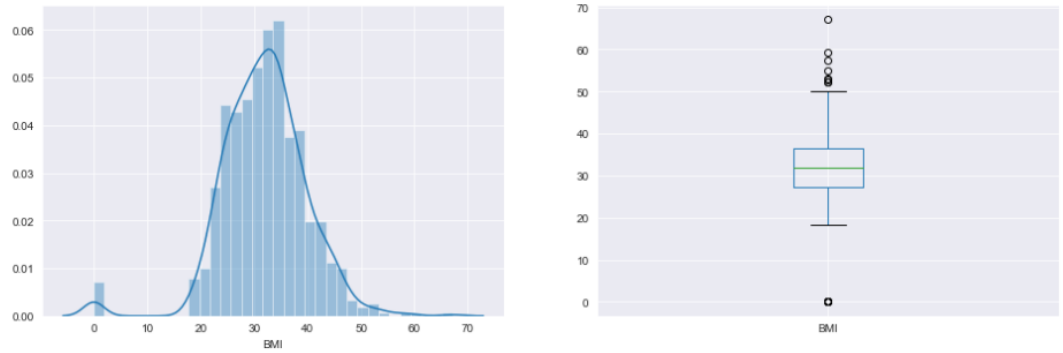
Distribution of pregnancies

```
plt.figure(2)
plt.subplot(121), sns.distplot(data['Pregnancies'])
plt.subplot(122), data['Pregnancies'].plot.box(figsize=(16,5))
plt.show()
```



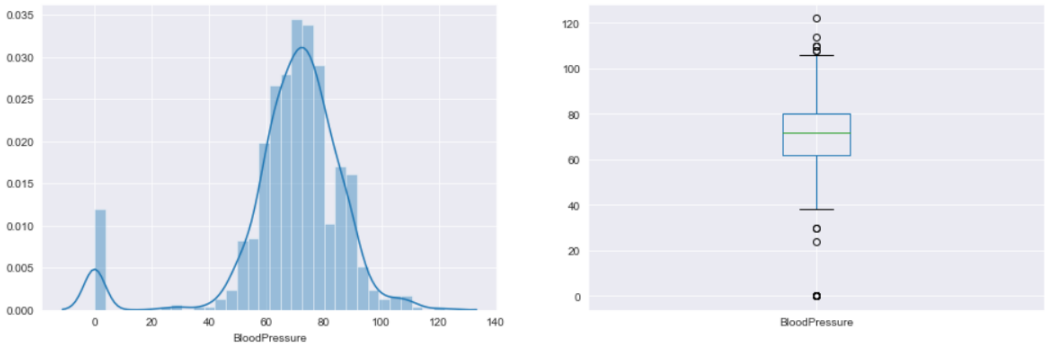
Distribution of BMI

```
plt.subplot(121), sns.distplot(data['BMI'])
plt.subplot(122), data['BMI'].plot.box(figsize=(16,5))
plt.show()
```



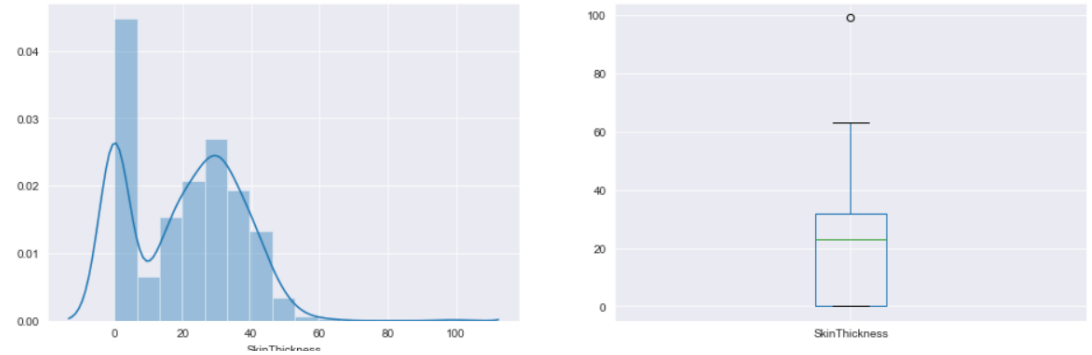
Distribution of Blood Pressure:

```
plt.subplot(121), sns.distplot(data['BloodPressure'])
plt.subplot(122), data['BloodPressure'].plot.box(figsize=(16,5))
plt.show()
```



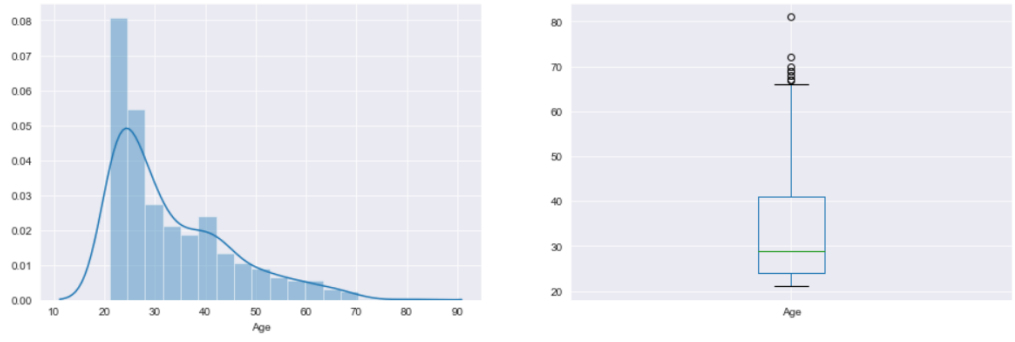
Distribution of Skin Thickness:

```
plt.subplot(121), sns.distplot(data['SkinThickness'])
plt.subplot(122), data['SkinThickness'].plot.box(figsize=(16,5))
plt.show()
```



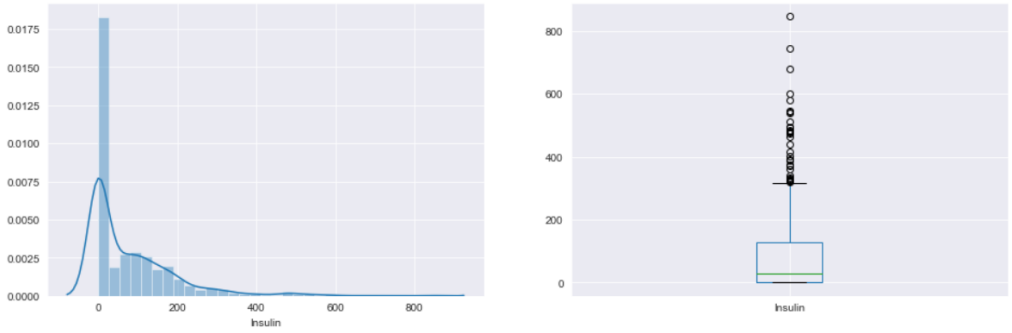
Distribution of Age:

```
plt.subplot(121), sns.distplot(data['Age'])
plt.subplot(122), data['Age'].plot.box(figsize=(16,5))
plt.show()
```



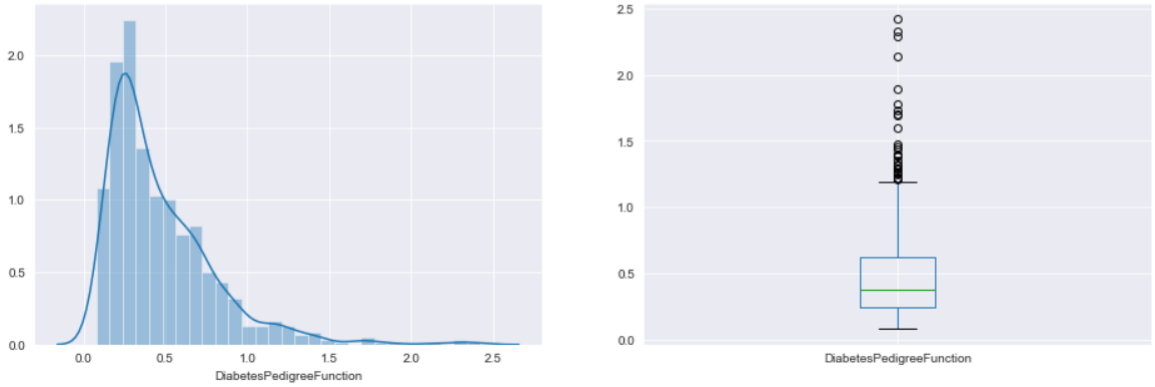
Distribution of Insulin:

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



Distribution of Diabetes Pedigree Function:

```
plt.subplot(121), sns.distplot(data['DiabetesPedigreeFunction'])
plt.subplot(122), data['DiabetesPedigreeFunction'].plot.box(figsize=(16,5))
plt.show()
```



Checking for any missing values in the data set:

```
#Check missing values

data.isnull().sum()
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

There are no missing values in the dataset. The dataset had already been cleaned.

Plotting relationships in the datasets:

There are different ways to display relationships using a dataset. You can use pair plots, joint plots, correlations, etc. we will the use pairplot to find out relationships in the dataset.

```
sns.pairplot(data)
```

```
<seaborn.axisgrid.PairGrid at 0x220dde58d30>
```

Next, we will proceed in checking the relationships by visualizing correlations as shown in the table below.

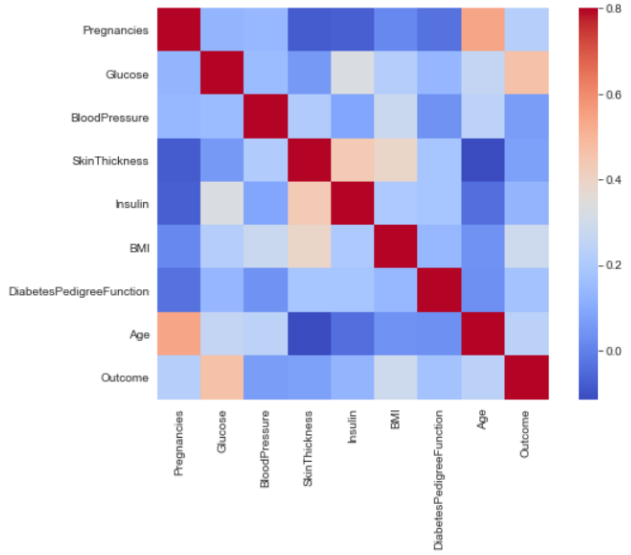
```
#check correlations

data.corr()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.221898
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.466581
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.065068
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.074752
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.130548
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.292695
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.173844
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.238356
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.000000

The table displays specific correlations for different variables in the dataset in probability form.

We can plot the correlations using a heatmap as shown below.

```
matrix = data.corr()
ax = plt.subplots(figsize=(9, 6)), sns.heatmap(matrix, vmax=.8, square=True, cmap="coolwarm")
```



Training the data:

We will now split our dataset before we train it. X will contain all the Independent variables while y will have the Dependent variable.

```
#Splitting the dataset

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

After successfully splitting the dataset, let us train it using train_test_split

```
from sklearn.model_selection import train_test_split

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

Before we build the model, let us impute the zero values in our dataset. If you check the head of the dataset, you will notice that there are some independent variables with zero values. This can make our model not efficient. We therefore, need to impute the zero values by using the mean of the other values in the same column. The code below shows how we can check the zero values in the dataset by printing for each variable.

```
#Check columns with zero values

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

We can notice that there are a total of 768 zero values in the X dataset (Independent variables). We have also been able to print the number of zero values for each column.

We will now use **mean** to impute the zero values as shown below. This method computes the mean of the column and imputes the values that have zero with the mean. This makes the dataset more meaningful for machine learning.

```
#Imputing zeros values in the dataset

from sklearn.preprocessing import Imputer

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

Building the model

As we stated earlier, we will use four models i.e. Random Forests, Decision Trees, XGBoost and Support Vector Machine to get the best accuracy score. ‘Accuracy’ metric is used to evaluate models. It is the ratio of the number of correctly predicted instances in a dataset divided by the total number of instances in the dataset. We will proceed further to explore more metrics to determine the best model.

Random Forests:

```
#Buildng the model using RandomForest

from sklearn.ensemble import RandomForestClassifier

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

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=200, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

```
predictions = rfc.predict(X_test)
```

```
#Getting the accuracy score for Random Forest

from sklearn import metrics

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

```
Accuracy_Score = 0.7559055118110236
```

Random Forest gives an accuracy_score of 0.7598

```
from sklearn.metrics import classification_report, confusion_matrix

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

```
[[132  30]
 [ 32  60]]
              precision    recall  f1-score   support

           0       0.80      0.81      0.81       162
           1       0.67      0.65      0.66        92

   micro avg       0.76      0.76      0.76       254
   macro avg       0.74      0.73      0.73       254
weighted avg       0.75      0.76      0.76       254
```

Decision Trees

```
#Building the model using DecisionTree

from sklearn.tree import DecisionTreeClassifier

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

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

```
predictions = dtree.predict(X_test)
```

```
#Getting the accuracy score for Decision Tree

from sklearn import metrics

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

```
Accuracy Score = 0.7440944881889764
```

As shown above, Decision Tree gives an accuracy_score of 0.7283.

```
from sklearn.metrics import classification_report, confusion_matrix

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

```
[[128  34]
 [ 31  61]]
              precision    recall  f1-score   support

           0       0.81      0.79      0.80       162
           1       0.64      0.66      0.65        92

   micro avg       0.74      0.74      0.74       254
   macro avg       0.72      0.73      0.72       254
weighted avg       0.75      0.74      0.74       254
```

XG Boost

```
#Building model using XGBoost

from xgboost import XGBClassifier

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

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=3, min_child_weight=1, missing=None,
       n_estimators=100, n_jobs=1, nthread=None,
       objective='binary:logistic', random_state=0, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=None,
       subsample=1, verbosity=1)
```

```
xgb_pred = xgb_model.predict(X_test)
```

```
#Getting accuracy score for XGBoost

from sklearn import metrics

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

```
Accuracy Score = 0.7795275590551181
```

XGBoost seems to be doing well with an accuracy score of 0.7795.

```
#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))
```

```
[[133  29]
 [ 27  65]]
              precision    recall  f1-score   support

           0       0.83      0.82      0.83       162
           1       0.69      0.71      0.70        92

   micro avg       0.78      0.78      0.78       254
   macro avg       0.76      0.76      0.76       254
weighted avg       0.78      0.78      0.78       254
```

Support Vector Machine:

```
#Building the model using Support Vector Machine (SVM)

from sklearn.svm import SVC

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

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

```
#Predict
svc_pred = svc_model.predict(X_test)
```

```
#Accuracy score for SVM
from sklearn import metrics

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

```
Accuracy Score = 0.6377952755905512
```

```
#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))
```

```
[[162   0]
 [ 92   0]]
              precision    recall  f1-score   support

           0       0.64      1.00      0.78       162
           1       0.00      0.00      0.00        92

   micro avg       0.64      0.64      0.64       254
   macro avg       0.32      0.50      0.39       254
weighted avg       0.41      0.64      0.50       254
```

Conclusion:

Therefore XGBoost is the best model for this prediction since it has an accuracy_score of 0.779.

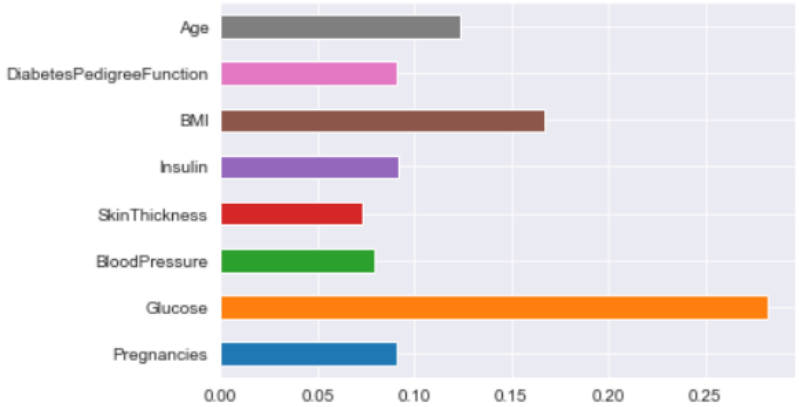
We will then look at the features that are most important when we use the XGBoost model for diabetes prediction.

```
#Getting feature importances
xgb_model.feature_importances_
```

```
array([0.09113621, 0.2825744 , 0.0796136 , 0.07278475, 0.09210336,
       0.16679356, 0.09089543, 0.12409869], dtype=float32)
```

```
#Plotting feature importances
(pd.Series(xgb_model.feature_importances_, index=X.columns)
   .plot(kind='barh'))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x220e3a58f98>
```



The graph shows that the most important feature in this diabetes prediction is **Glucose** followed by **BMI**.

Predictions:

Finally, let us use XGBoost Model to predict the possibility of a patient having Diabetes or not (1 or 0). The following are the prediction probabilities of absence or presence of Diabetes respectively.

```
#Printing prediction probabilities for the test data
print('Prediction Probabilities')
xgb_model.predict_proba(X_test)
```

```
Prediction Probabilities

array([[0.9815974 , 0.0184026 ],
       [0.08197236, 0.91802764],
       [0.505653  , 0.494347  ],
       [0.97009706, 0.02990292],
       [0.32111228, 0.6788877 ],
       [0.28722966, 0.71277034],
       [0.95970166, 0.04029833],
       [0.9591639 , 0.0408361 ],
       [0.14804524, 0.85195476],
       [0.58868796, 0.41131204],
       [0.03294134, 0.96705866],
       [0.992144  , 0.00785601],
       [0.06900525, 0.93099475],
       [0.07198477, 0.92801523],
       [0.8095031 , 0.19049695],
       [0.8651936 , 0.13480641],
       [0.874498  , 0.12550198],
```

We can see that the patient at index 0 has a 98.1% chance of absence of diabetes, while the patient at index 1 has a 91.8% predicted chance of having diabetes.