Group Members and IDs

RDS2_Group3

1. Chin Jun Wai (20WMR08862)
2. Tan Kai Yuan (20WMR08886)
3. Kow Yee Hui (20WMR08868)

1. Business Understanding

1.1 Determine Business Objectives

1.1.1 Background

Diabetes is a common disease worldwide including in Malaysia. This disease can be suffered by anyone no matter aged, adult or young. It can be due to a person consuming too much food and drinks which are in high sugar levels in his or her daily life. When consumed too much sugar, it would cause a person's pancreas to malfunction from time to time due to overload with digesting the glucose. Therefore, it causes a person to get diabetes with high chance due to insufficient insulin produced by the pancreas to digest the sugar we consume. According to CodeBlue (2019), the Minister of Health has estimated the cost of treating chronic diabetes and high blood pressure patients at government facilities which is RM 3.2 billion for the year of 2019. Those costs included lab tests, consultations, infrastructure and so on (CodeBlue 2019). Furthermore, “17.5 per cent of the Malaysian population, or an estimated 3.5 million adults, suffered from diabetes” (CodeBlue 2019).

1.1.2 Business Objectives

With the information that we have mentioned, we have decided to propose our goal which is helping hospitals to classify their patients whether they have diabetes or not because according to research, the earlier we get a result of diabetes diagnosis from a patient which is positive, the better the doctors are able to handle it. In addition, we would like to help the hospital to reduce the cost of doing the lab tests since research has mentioned that it is around RM 3.2 billion for treating diabetes and high blood pressure patients. Therefore, we have the idea of doing machine learning on diabetes prediction so that most of the hospitals can implement it and have better control over their diabetes patients.

1.2 Determine data mining goals

1. Business success criteria

To increase productivity of a hospital in predicting whether a patient has diabetes accurately, at the same time reducing the rate of “machine predicted a patient is healthy but end up the patient has diabetes”.

2. Data mining success criteria

Our criteria is to let the machine learning model have a result of high accuracy and try to reduce false negative that contributes to the type II error.

2. Data Understanding

2.1 Data Loading

Import libraries

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

Import dataset

The dataset used is from the National Institute of Diabetes and Digestive and Kidney Diseases. We take this dataset from kaggle which is a world's data science community. Kaggle faced some limitations when collecting these data, because all the patients in the data are Pima Indian females at least 21 years old.

In [2]:

```
df = pd.read_csv("diabetes.csv")
df
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunct
0	6	148	72	35	0	33.6	0.
1	1	85	66	29	0	26.6	0.
2	8	183	64	0	0	23.3	0.
3	1	89	66	23	94	28.1	0.
4	0	137	40	35	168	43.1	2.
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	0.
764	2	122	70	27	0	36.8	0.
765	5	121	72	23	112	26.2	0.
766	1	126	60	0	0	30.1	0.
767	1	93	70	31	0	30.4	0.

768 rows × 9 columns

The diabetes dataset included 768 rows and 9 columns. It consists of one target variable and 8 medical predictor variables. After we validate the dataset, we also know that this diabetes dataset can fulfill our requirements because it is complete, clear and consistent.

Variable :

1. Outcome - 0 is non-diabetes, 1 is diabetes
2. Pregnancies - Number of times pregnant
3. Glucose - Plasma glucose concentration
4. BloodPressure - Diastolic blood pressure
5. SkinThickness - Triceps skinfold thickness
6. Insulin - insulin level
7. BMI - Body Mass Index
8. DiabetesPedigreeFunction - Diabetes Pedigree Function
9. Age - Age of patient (at least 21 years old)

In [3]:

```
df.shape
```

Out[3]:

```
(768, 9)
```

Splitting dataset into train dataset and test dataset

In [4]:

```
X = df.drop("Outcome", axis = 1)
y = df['Outcome']
X.shape, y.shape
```

Out[4]:

```
((768, 8), (768,))
```

In [5]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 4
X_train.shape, X_test.shape
```

Out[5]:

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

In [6]:

```
train_data = pd.concat([X_train, y_train] ,axis =1)
test_data = pd.concat([X_test, y_test], axis =1)
train_data.shape, test_data.shape
```

Out[6]:

```
((614, 9), (154, 9))
```

In [7]:

```
train_data.describe()
```

Out[7]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabet
count	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	
mean	3.742671	120.855049	69.415309	20.399023	81.438111	31.983388	
std	3.313264	32.035057	18.512599	15.433974	116.234835	7.740625	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	100.000000	64.000000	0.000000	0.000000	27.100000	
50%	3.000000	117.000000	72.000000	23.000000	42.500000	32.000000	
75%	6.000000	139.000000	80.000000	32.000000	129.750000	36.375000	
max	17.000000	199.000000	122.000000	63.000000	846.000000	67.100000	

In [8]:

```
train_data.columns
```

Out[8]:

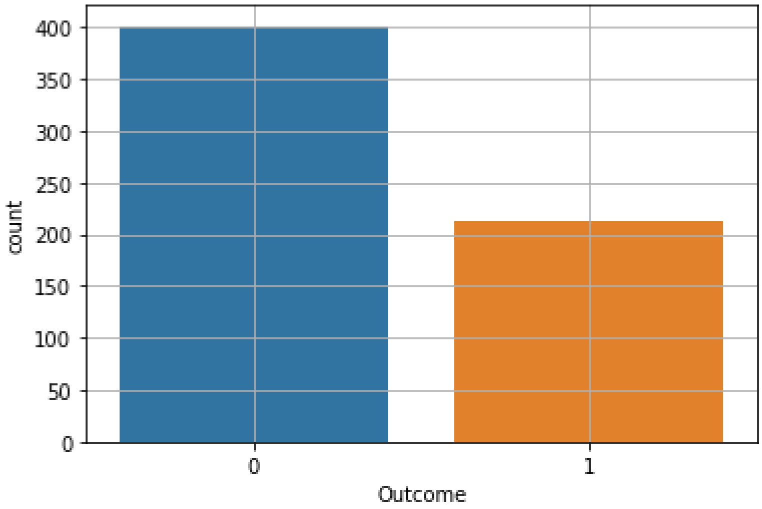
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insuli
n',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

2.2 Data Exploration and Visualization on Training Dataset

In this section, we have conducted the data exploration. In order to get the whole pattern of the main points, we explore a large dataset by an unstructured way such as data distribution and box plot. Data distribution allows us to easily know all the possible values of the data and tells us how often these data occur.

In [9]:

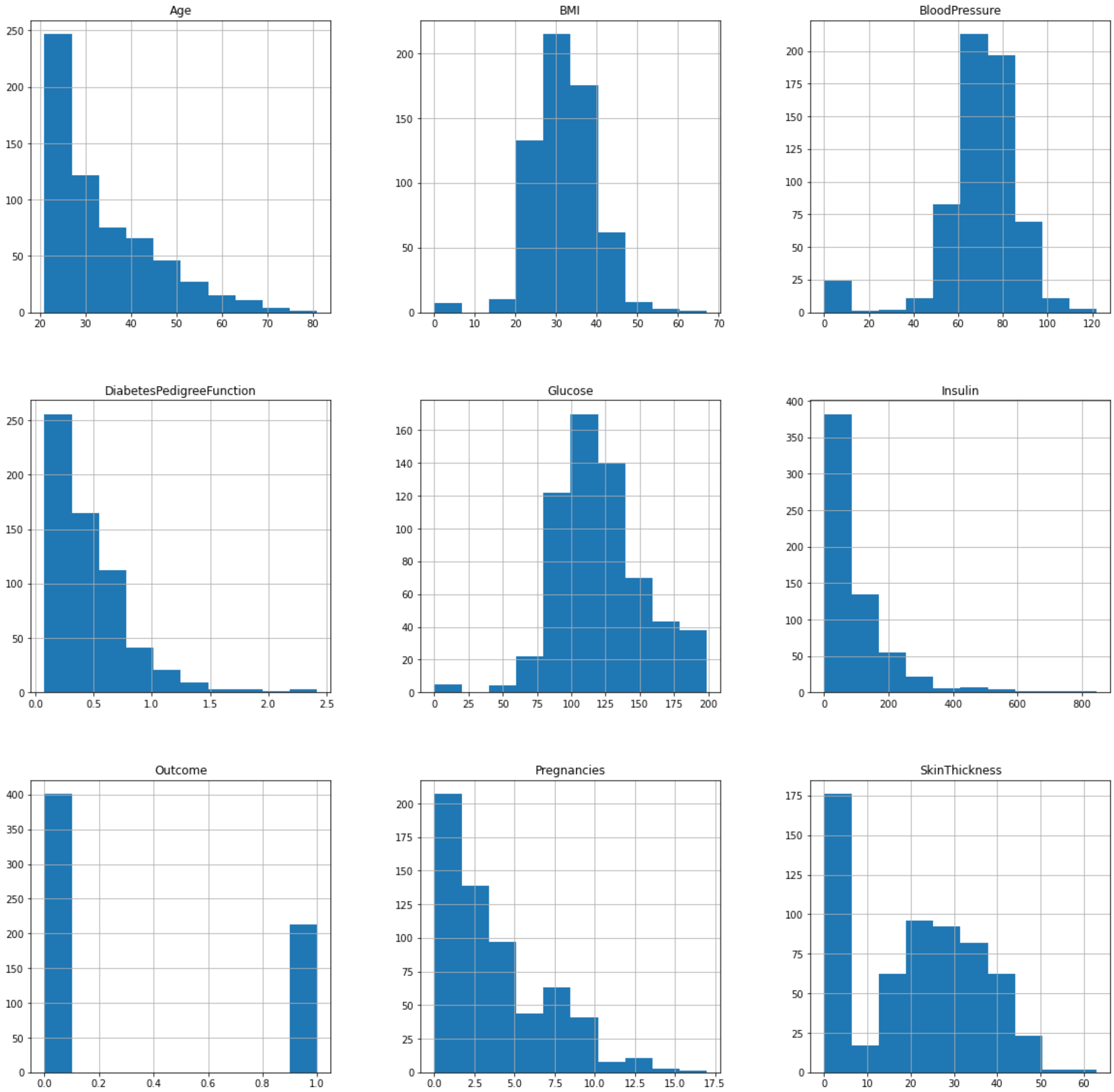
```
sns.countplot(train_data.Outcome)
plt.grid(True)
```



From the plot above, it shows more patient is non-diabetes.

In [10]:

```
p = train_data.hist(figsize = (20,20))
```



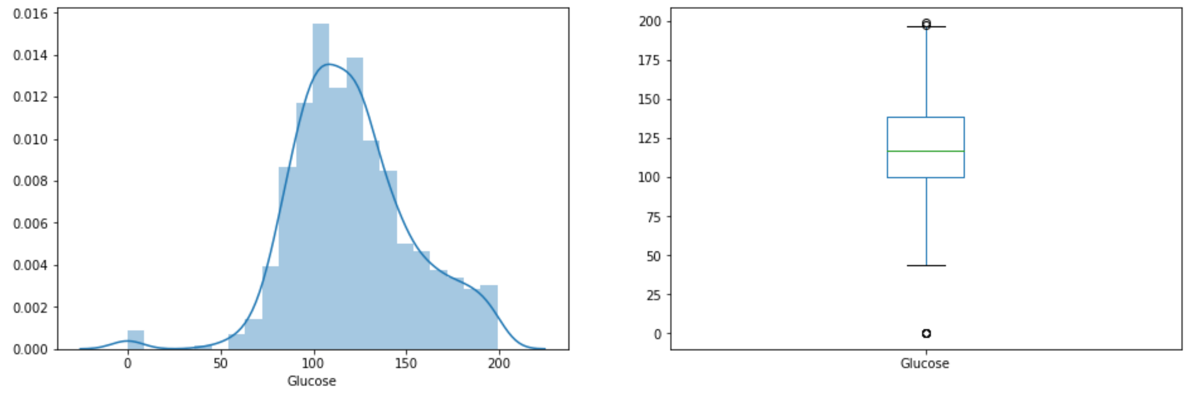
1. From Distribution of Age, it shows that most of the data is comes from age between 21 to 27. Furthermore, it means that majorities are young adults.

2. From Distribution of BMI, it shows that few outliers in diabetes dataset. Some of the participants's BMI is out of expected range between 18 to 25. It means they are obese.
3. From Distribution of BloodPressure, it shows that the distribution looks normal. It means almost all of the participants have normal blood pressure.
4. From Distribution of Diabetes Pedigree Function, it shows that Diabetes Pedigree Function has an exponential distribution with Outcome variable.
5. From Distribution of Glucose, it shows that the data are slightly skewed to right. It means the glucose level of almost 60% participants is higher.
6. From Distribution of Insulin, it shows that serum insulin in most of the participants are low. But it also shows a lot of outliers in the diabetes dataset. In case the expected range is between 16 to 166, they have high serum insulin.
7. From Distribution of Pregnancies, it shows that the data is right skewed, It means more of the participants are zero of pregnancy. This is because some participants in this dataset are more than 21 years old so many of them are unmarried.
8. From Distribution of Skin Thickness, it shows that the data of skin thickness are skewed to the right a bit. In case of the expected skin thickness is low, thus it means less participants have high skin thickness.

Distribution of Glucose

In [11]:

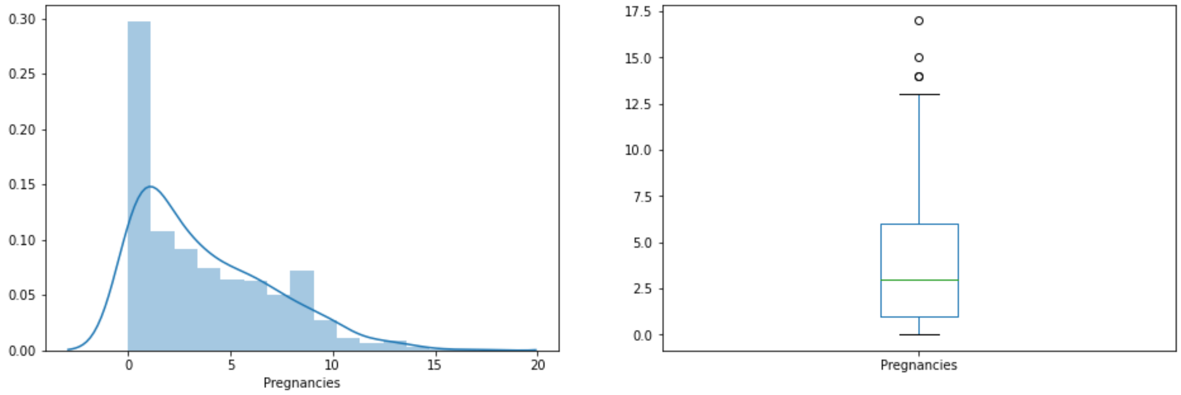
```
#Glucose
plt.figure(1)
plt.subplot(121),sns.distplot(train_data['Glucose'])
plt.subplot(122),train_data['Glucose'].plot.box(figsize=(16,5))
plt.show()
```



Distribution of Pregnancies

In [12]:

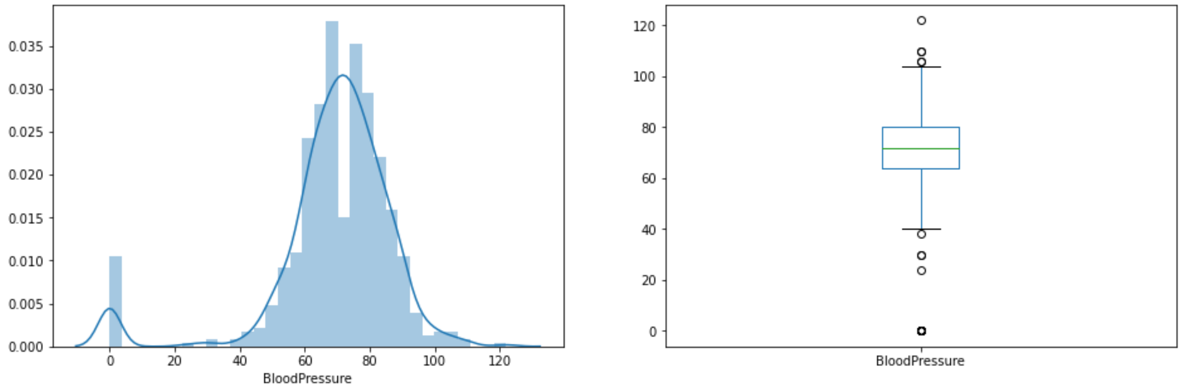
```
#Pregnancies
plt.figure(1)
plt.subplot(121),sns.distplot(train_data['Pregnancies'])
plt.subplot(122),train_data['Pregnancies'].plot.box(figsize=(16,5))
plt.show()
```



Distribution of BloodPressure

In [13]:

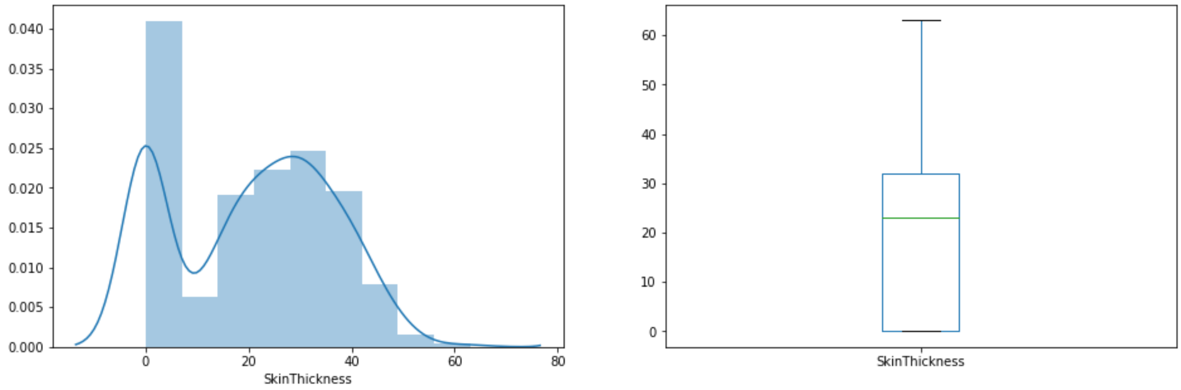
```
#BloodPressure
plt.figure(1)
plt.subplot(121),sns.distplot(train_data['BloodPressure'])
plt.subplot(122),train_data['BloodPressure'].plot.box(figsize=(16,5))
plt.show()
```



Distribution of SkinThickness

In [14]:

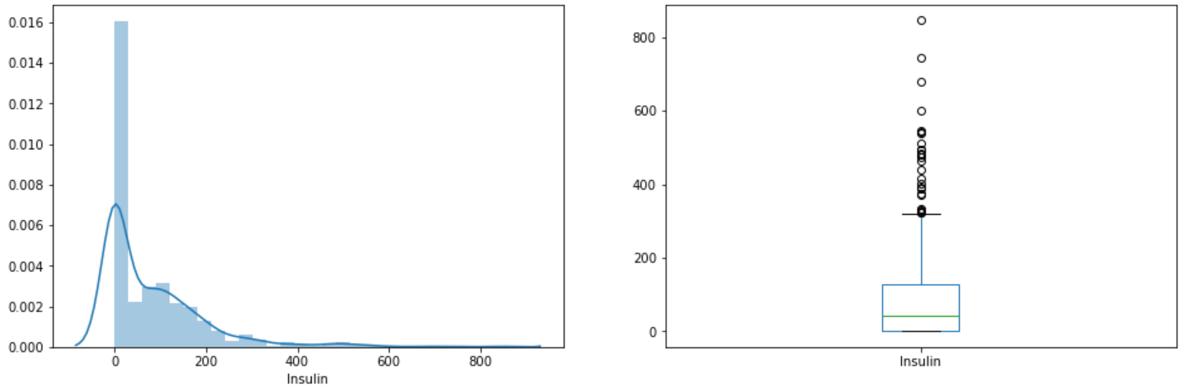
```
#SkinThickness
plt.figure(1)
plt.subplot(121),sns.distplot(train_data['SkinThickness'])
plt.subplot(122),train_data['SkinThickness'].plot.box(figsize=(16,5))
plt.show()
```



Distribution of Insulin

In [15]:

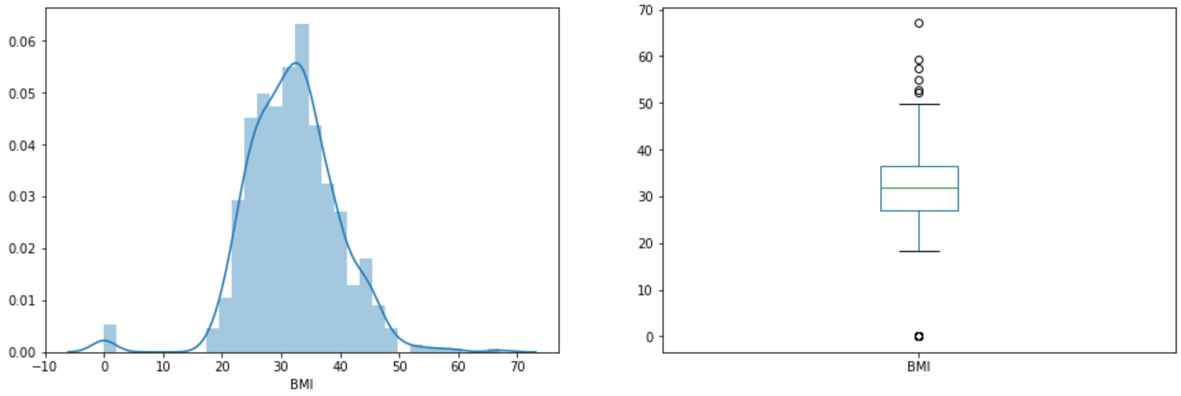
```
#Insulin
plt.figure(1)
plt.subplot(121),sns.distplot(train_data['Insulin'])
plt.subplot(122),train_data['Insulin'].plot.box(figsize=(16,5))
plt.show()
```



Distribution of BMI

In [16]:

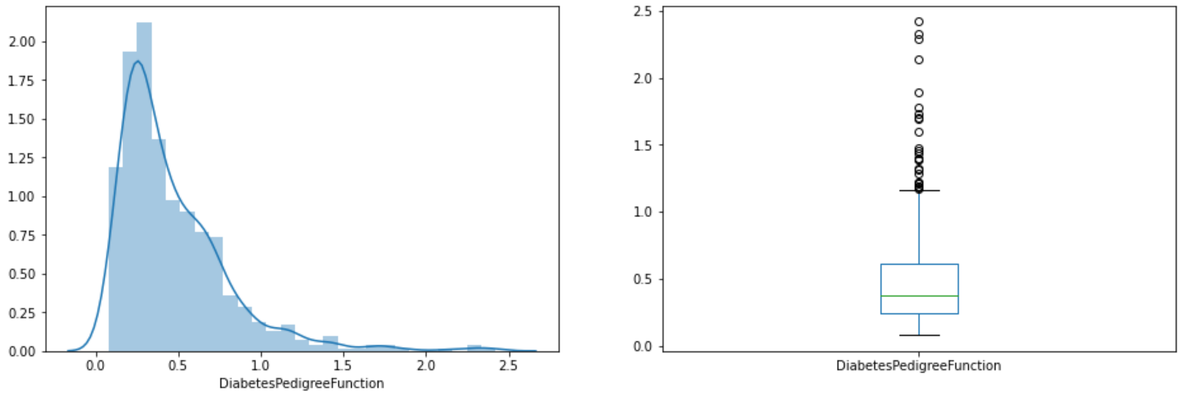
```
#BMI
plt.figure(1)
plt.subplot(121),sns.distplot(train_data['BMI'])
plt.subplot(122),train_data['BMI'].plot.box(figsize=(16,5))
plt.show()
```



Distribution of DiabetesPedigreeFunction

In [17]:

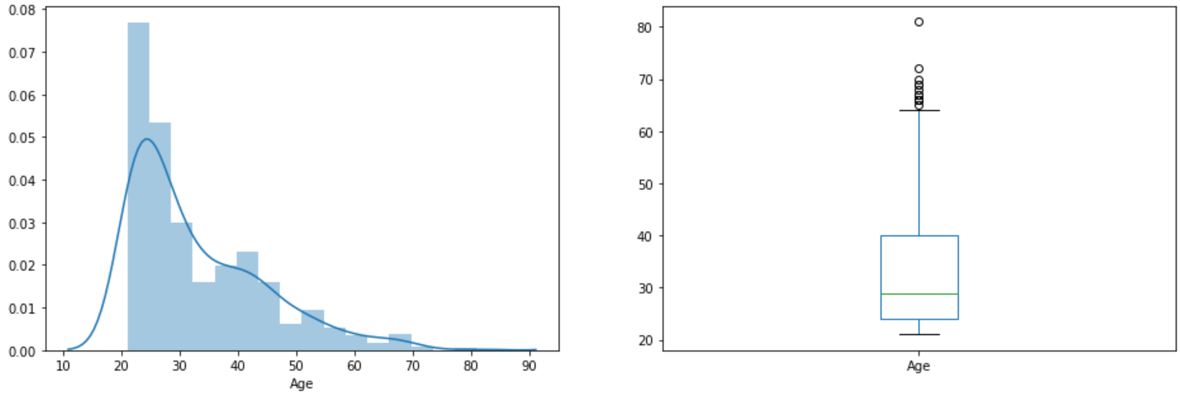
```
#DiabetesPedigreeFunction
plt.figure(1)
plt.subplot(121),sns.distplot(train_data['DiabetesPedigreeFunction'])
plt.subplot(122),train_data['DiabetesPedigreeFunction'].plot.box(figsize=(16,5))
plt.show()
```



Distribution of Age

In [18]:

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



Relationships between pairs or small numbers of attributes

Pair plots to find out the relationship in the dataset

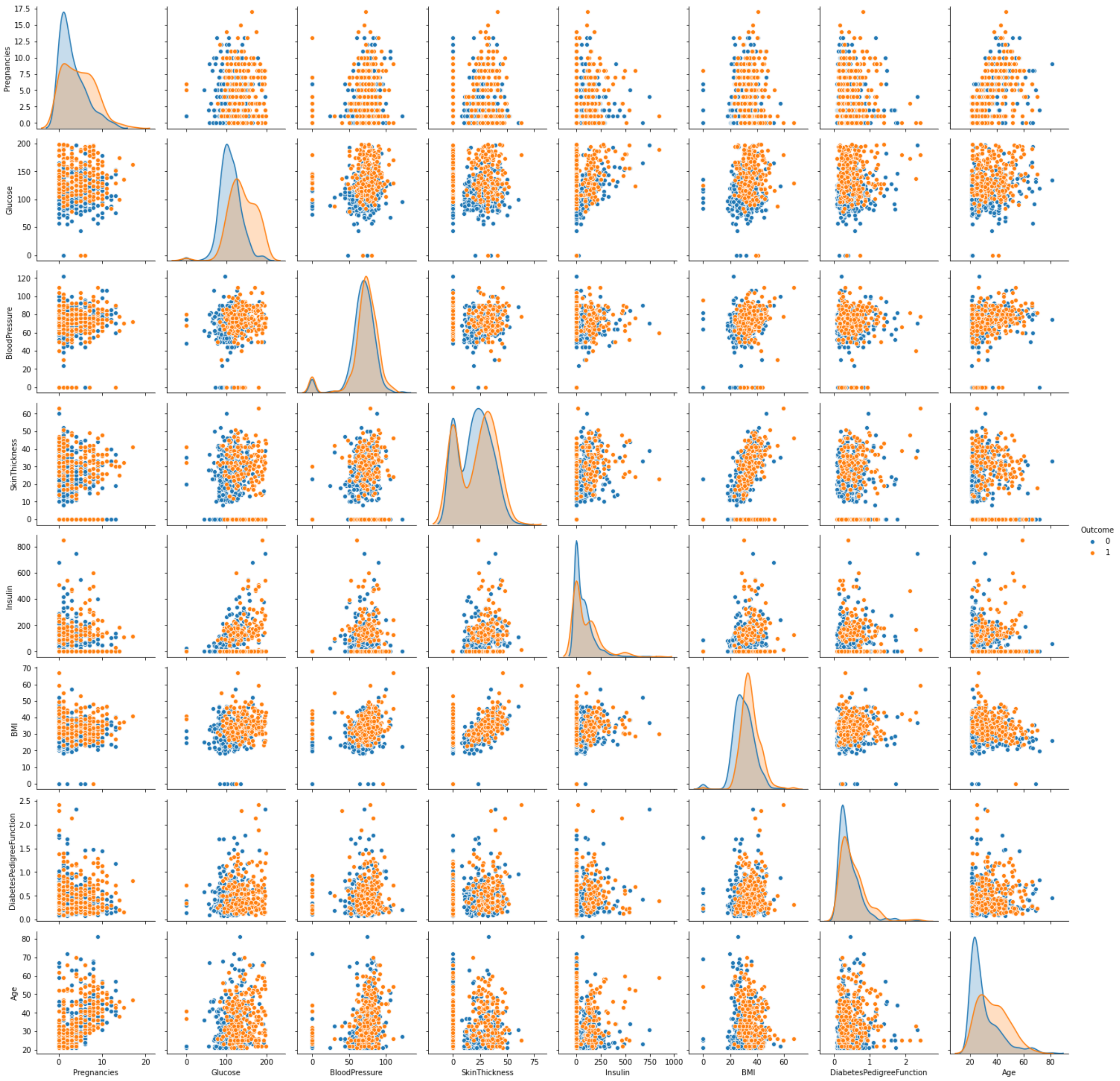
From Scatter Plots, it shows that the relationship between each attribution taken pairwise. Based on the Scatter Plots, we can see that almost no attributes are able to clearly seperate the two outcome. Only BloodPressure and Glucose seen have positive linear relationship. Furthermore, there is also a positive relationship between BMI and skinthickness.

In [19]:

```
sns.pairplot(train_data, hue = "Outcome")
```

Out[19]:

```
<seaborn.axisgrid.PairGrid at 0x2205f50ad90>
```



Check the relationship by visualizing the correlations between the variables

In [20]:

```
train_data.corr()
```

Out[20]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	
Pregnancies	1.000000	0.142507	0.157544	-0.070691	-0.066401	0.0
Glucose	0.142507	1.000000	0.158320	0.017320	0.337064	0.2
BloodPressure	0.157544	0.158320	1.000000	0.178062	0.085834	0.2
SkinThickness	-0.070691	0.017320	0.178062	1.000000	0.437564	0.3
Insulin	-0.066401	0.337064	0.085834	0.437564	1.000000	0.1
BMI	0.027197	0.205162	0.250012	0.386223	0.196035	1.0
DiabetesPedigreeFunction	-0.018944	0.141597	0.044826	0.174623	0.158923	0.1
Age	0.553048	0.270740	0.239588	-0.149863	-0.043823	0.0
Outcome	0.207550	0.456117	0.082046	0.057912	0.108498	0.3

Plot the correlations by using the heatmap

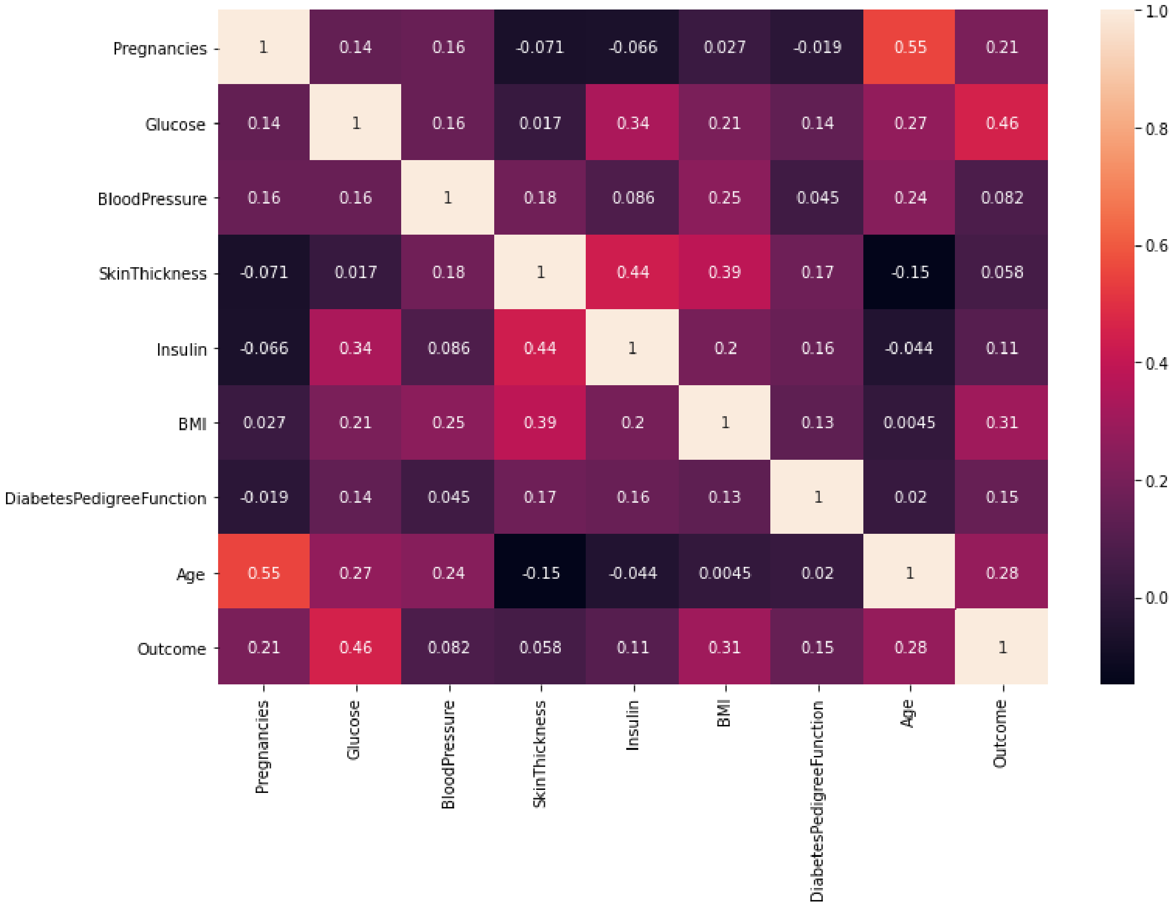
We plot a heatmap to observe which variables have a strong relationship with diabetes. For the Heatmap, if the variable is brighter color, it indicates that this variable has a strong relationship with diabetes. Heatmap shows that "Glucose" has highest correlation with the Outcome variable. Furthermore, "Age" has a moderate positive relationship with "Pregnancies". Beisdes that, "Insulin", "Diabetes Pedigree Function" and "Blood Pressure" have quite low correlation with the outcome variable while the "Skin Thickness" is the lowest.

In [21]:

```
corr = train_data.corr()
#print(corr)
fig, ax = plt.subplots(figsize=(12,8))
sns.heatmap(corr, annot= True,
         xticklabels=corr.columns,
         yticklabels=corr.columns)

#brighter color indicates more correlation
```

Out[21]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x22061f7d490>
```



3. Data Preparation

3.1 Missing Value and Invalid Value

Data quality will cause a large impact on the result of a data analysis. If there are missing values or incorrect data in a dataset, these can result in incorrect experimental results. First of all, we have checked for missing value in the dataset and we found that there are no missing value in our dataset. However, we have done research and found that the features like “Glucose”, “Blood Pressure”, “Skin Thickness”, “Insulin”, and "BMI" should not in zero value so there are invalid zero value in these features. Therefore, we need to replace the zero value in these features with NaN and fill up the NaN with median value of the respective column. After impute, we describe the data again to have an overview of data.

Check For Any Missing Value In Dataset

In [22]:

```
# check null values
train_data.isnull().sum()
```

Out[22]:

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

Replace Invalid Value with NaN

In [23]:

```
# replace 0 with NaN
train_data[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = train_data[
    ['Glucose','BloodPressure','SkinThickness','Insulin','BMI']].replace(0,np.NaN)

test_data[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = test_data[
    ['Glucose','BloodPressure','SkinThickness','Insulin','BMI']].replace(0,np.NaN)

print (train_data.isnull().sum())
print (test_data.isnull().sum())
```

```
Pregnancies                   0
Glucose                       5
BloodPressure                24
SkinThickness               176
Insulin                     290
BMI                           7
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
Pregnancies                  0
Glucose                      0
BloodPressure               11
SkinThickness               51
Insulin                     84
BMI                          4
DiabetesPedigreeFunction     0
Age                          0
Outcome                      0
dtype: int64
```

Replace NaN with Median Value

In [24]:

```
# replace null value with suitable data
train_data['Glucose'].fillna(train_data['Glucose'].median(), inplace = True)
train_data['BloodPressure'].fillna(train_data['BloodPressure'].median(), inplace = True)
train_data['SkinThickness'].fillna(train_data['SkinThickness'].median(), inplace = True)
train_data['Insulin'].fillna(train_data['Insulin'].median(), inplace = True)
train_data['BMI'].fillna(train_data['BMI'].median(), inplace = True)

test_data['Glucose'].fillna(train_data['Glucose'].median(), inplace = True)
test_data['BloodPressure'].fillna(train_data['BloodPressure'].median(), inplace = True)
test_data['SkinThickness'].fillna(train_data['SkinThickness'].median(), inplace = True)
test_data['Insulin'].fillna(train_data['Insulin'].median(), inplace = True)
test_data['BMI'].fillna(train_data['BMI'].median(), inplace = True)
```

Describe the data after imputation

In [25]:

```
train_data.describe()
```

Out[25]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Dia
count	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	
mean	3.742671	121.815961	72.229642	28.568404	138.115635	32.348208	
std	3.313264	30.104012	12.099278	8.410029	88.650282	6.935618	
min	0.000000	44.000000	24.000000	8.000000	14.000000	18.200000	
25%	1.000000	100.000000	64.000000	24.000000	116.000000	27.325000	
50%	3.000000	118.000000	72.000000	28.500000	120.000000	32.000000	
75%	6.000000	139.000000	80.000000	32.000000	129.750000	36.375000	
max	17.000000	199.000000	122.000000	63.000000	846.000000	67.100000	

3.2 Balancing the Data

From the data visualization, we can observe that this dataset is imbalanced bacause the False outcome (No Diabetes) is more than True outcome (Have Diabetes). Therefore, we need to balance the data in order to let the machine learning model to predict the patient on an equal classification rate such that patient is always to be predicted to have 50 % chance to get diabetes or 50% chance not to get diabetes. If we do not balance the data, the model will skew to one of the sides and tend to predict a patient do not have diabetes.

Split the train test data back to XY

Before we balance the data, we firstly spilt the train test data back to X_train, X_test, y_train and y_test.

In [26]:

```
X_train = train_data.drop("Outcome", axis = 1)
y_train = train_data['Outcome']

X_test = test_data.drop("Outcome", axis = 1)
y_test = test_data['Outcome']


X_train_drop10 = train_data.drop(["Pregnancies","BloodPressure","SkinThickness","Outcome"],
y_train_drop10 = train_data['Outcome']

X_test_drop10 = test_data.drop(["Pregnancies","BloodPressure","SkinThickness","Outcome"], a
y_test_drop10 = test_data['Outcome']

names = X_train.columns

X_train_drop10.shape , X_test_drop10.shape
```

Out[26]:

```
((614, 5), (154, 5))
```

Balance the data

We have used SMOTE, one of the Oversampling method to deal with the imbalance data. SMOTE will detect the class which has lower number of rows and create more rows of the class until it gets the same number with the class that has higher number of rows.

```
In [27]:
```

```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state = 42)
X_train, y_train = sm.fit_sample(X_train, y_train)
X_train_drop10, y_train_drop10 = sm.fit_sample(X_train_drop10, y_train_drop10)
print (X_train.shape, y_train.shape)
print (X_train_drop10.shape, y_train_drop10.shape)
```

```
(802, 8) (802,)
(802, 5) (802,)
```

3.3 Standardizing the Data

Before applying the algorithm onto data, we need to standardize the data with Standard Scaler, because the data might behave badly if the individual features do not more or less look like standard normally distributed data. The standardize will remove the mean and scale to unit variance so that the data distribution will have a mean value 0 and standard deviation of 1.

Standardize the data

```
In [28]:
```

```
#standardize
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

X_train_drop10 = sc.fit_transform(X_train_drop10)
X_test_drop10 = sc.transform(X_test_drop10)
```

4.Modelling

K folds

In [29]:

```
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import cross_val_score
```

Cross validation for Random Forest Classifier

In [30]:

```

rfc_cross = cross_val_score(estimator=RandomForestClassifier(random_state = 42),
                                 X=X_train, y=y_train)
rfc_cross
```

Out[30]:

```
array([0.78881988, 0.81987578, 0.7875    , 0.85625   , 0.86875   ])
```

In [31]:

```
print('Mean Accuracy:{}'.format(rfc_cross.mean()))
```

```
Mean Accuracy:0.8242391304347827
```

Cross validation for SVC

In [32]:

```
svc_cross = cross_val_score(estimator=SVC(random_state = 42),
                                 X=X_train, y=y_train)
svc_cross
```

Out[32]:

```
array([0.77639752, 0.80745342, 0.7625    , 0.80625   , 0.83125   ])
```

In [33]:

```
print('Mean Accuracy:{}'.format(svc_cross.mean()))
```

```
Mean Accuracy:0.7967701863354038
```

Cross validation for Logistic Regression

In [34]:

```
lgs_cross = cross_val_score(estimator= LogisticRegression(random_state = 42),
                                 X=X_train, y=y_train)
lgs_cross
```

Out[34]:

```
array([0.74534161, 0.7826087 , 0.71875   , 0.7375    , 0.7875    ])
```

In [35]:

```
print('Mean Accuracy:{}'.format(lgs_cross.mean()))
```

```
Mean Accuracy:0.7543400621118013
```

Cross validation for Neural Network

In [36]:

```
nn_cross = cross_val_score(estimator= MLPClassifier(random_state = 42),
                                 X=X_train, y=y_train)
nn_cross
```

```
C:\Users\ASUS\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer
_perceptron.py:582: ConvergenceWarning: Stochastic Optimizer: Maximum iterat
ions (200) reached and the optimization hasn't converged yet.
  warnings.warn(
C:\Users\ASUS\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer
_perceptron.py:582: ConvergenceWarning: Stochastic Optimizer: Maximum iterat
ions (200) reached and the optimization hasn't converged yet.
  warnings.warn(
C:\Users\ASUS\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer
_perceptron.py:582: ConvergenceWarning: Stochastic Optimizer: Maximum iterat
ions (200) reached and the optimization hasn't converged yet.
  warnings.warn(
C:\Users\ASUS\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer
_perceptron.py:582: ConvergenceWarning: Stochastic Optimizer: Maximum iterat
ions (200) reached and the optimization hasn't converged yet.
  warnings.warn(
C:\Users\ASUS\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer
_perceptron.py:582: ConvergenceWarning: Stochastic Optimizer: Maximum iterat
ions (200) reached and the optimization hasn't converged yet.
  warnings.warn(
```

Out[36]:

```
array([0.7826087, 0.8136646, 0.7875   , 0.8375   , 0.85625  ])
```

In [37]:

```
print('Mean Accuracy:{}'.format(nn_cross.mean()))
```

```
Mean Accuracy:0.8155046583850931
```

Cross validation for KNN

In [38]:

```
knn_cross = cross_val_score(estimator= KNeighborsClassifier(),
                                 X=X_train, y=y_train)
knn_cross
```

Out[38]:

```
array([0.73291925, 0.75776398, 0.775     , 0.8125    , 0.8625    ])
```

In [39]:

```
print('Mean Accuracy:{}'.format(knn_cross.mean()))
```

```
Mean Accuracy:0.788136645962733
```

Cross validation for Naive Baiyes

In [40]:

```
nb_cross = cross_val_score(estimator= GaussianNB(),
                                 X=X_train, y=y_train)
nb_cross
```

Out[40]:

```
array([0.70807453, 0.73291925, 0.75625   , 0.7125    , 0.73125   ])
```

In [41]:

```
print('Mean Accuracy:{}'.format(nb_cross.mean()))
```

```
Mean Accuracy:0.7281987577639752
```

Cross validation for XGboost

In [42]:

```
xgb_cross = cross_val_score(estimator= XGBClassifier(random_state = 42),
                                 X=X_train, y=y_train)
xgb_cross
```

Out[42]:

```
array([0.77639752, 0.76397516, 0.8       , 0.86875   , 0.875     ])
```

In [43]:

```
print('Mean Accuracy:{}'.format(xgb_cross.mean()))
```

```
Mean Accuracy:0.8168245341614906
```

Comparing mean training accuracy without setting parameters

In [44]:

```
print("Random Forest Classifier      : {} %".format(rfc_cross.mean() * 100))
print("Support Vector Classifier     : {} %".format(svc_cross.mean() * 100))
print("Logistic Regression Classifier: {} %".format(lgs_cross.mean() * 100))
print("Neural Network Classifier     : {} %".format(nn_cross.mean() * 100))
print("KNN Classifier                : {} %".format(knn_cross.mean() * 100))
print("Naive Baiyes Classifier       : {} %".format(nb_cross.mean() * 100))
print("XGboost Classifier            : {} %".format(xgb_cross.mean() * 100))
```

```
Random Forest Classifier      : 82.42391304347827 %
Support Vector Classifier     : 79.67701863354037 %
Logistic Regression Classifier: 75.43400621118013 %
Neural Network Classifier     : 81.55046583850931 %
KNN Classifier                : 78.8136645962733 %
Naive Baiyes Classifier       : 72.81987577639752 %
XGboost Classifier            : 81.68245341614906 %
```

Based on the results above, we can roughly see that Random Forest Classifier having the highest mean training accuracy among the classifiers and followed by XGboost classifier and Neural Network classfier. Therefore, our main focus will be on Random Forest Classifier.

Random Forest

In [45]:

```
from sklearn.ensemble import RandomForestClassifier

RandomForestC = RandomForestClassifier(n_estimators= 300, random_state=42 , criterion = "en
RandomForestC.fit(X_train,y_train)
predictions = RandomForestC.predict(X_test)
print(classification_report(y_test, predictions))
print(confusion_matrix(y_test,predictions))
print("Accuracy for training: {}".format(RandomForestC.score(X_train, y_train)))
print("Accuracy for testing: {}\n\n".format(RandomForestC.score(X_test, y_test)))
```

```
              precision    recall  f1-score   support

           0       0.89      0.78      0.83        99
           1       0.67      0.82      0.74        55

    accuracy                           0.79       154
   macro avg       0.78      0.80      0.78       154
weighted avg       0.81      0.79      0.80       154

[[77 22]
 [10 45]]
Accuracy for training: 1.0
Accuracy for testing: 0.7922077922077922
```

Feature Importance

In [46]:

```
for x in range(len(RandomForestC.feature_importances_)):
    print("{}: {} %".format(names[x],np.round(RandomForestC.feature_importances_[x] * 100 )
```

```
Pregnancies: 7.0 %
Glucose: 24.0 %
BloodPressure: 8.0 %
SkinThickness: 8.0 %
Insulin: 10.0 %
BMI: 17.0 %
DiabetesPedigreeFunction: 12.0 %
Age: 14.0 %
```

In [47]:

```
RandomForestC2 = RandomForestClassifier(n_estimators= 300, random_state=42 , criterion = "e
RandomForestC2.fit(X_train_drop10,y_train_drop10)
predictions2 = RandomForestC2.predict(X_test_drop10)
print(classification_report(y_test_drop10, predictions2))
print(confusion_matrix(y_test_drop10,predictions))
print("Accuracy for training: {}".format(RandomForestC2.score(X_train_drop10, y_train_drop1
print("Accuracy for testing: {}\n\n".format(RandomForestC2.score(X_test_drop10, y_test_drop
```

```
              precision    recall  f1-score   support

           0       0.83      0.76      0.79        99
           1       0.62      0.73      0.67        55

    accuracy                           0.75       154
   macro avg       0.73      0.74      0.73       154
weighted avg       0.76      0.75      0.75       154

[[77 22]
 [10 45]]
Accuracy for training: 1.0
Accuracy for testing: 0.7467532467532467
```

Grid Search for random forest classifier

In [48]:

```
from sklearn.model_selection import GridSearchCV

parameters = { 'n_estimators': [100, 300, 500, 800, 1000],
    'criterion': ['gini', 'entropy'],
    'bootstrap': [True, False]  }

gcv = GridSearchCV(
    estimator = RandomForestC ,
    param_grid = parameters,
    scoring = 'accuracy',
    cv = 10,
    n_jobs = -1
    )

gcv = gcv.fit(X_train, y_train)
gcv.best_score_
```

Out[48]:

```
0.844182098765432
```

In [49]:

```
gcv.best_params_
```

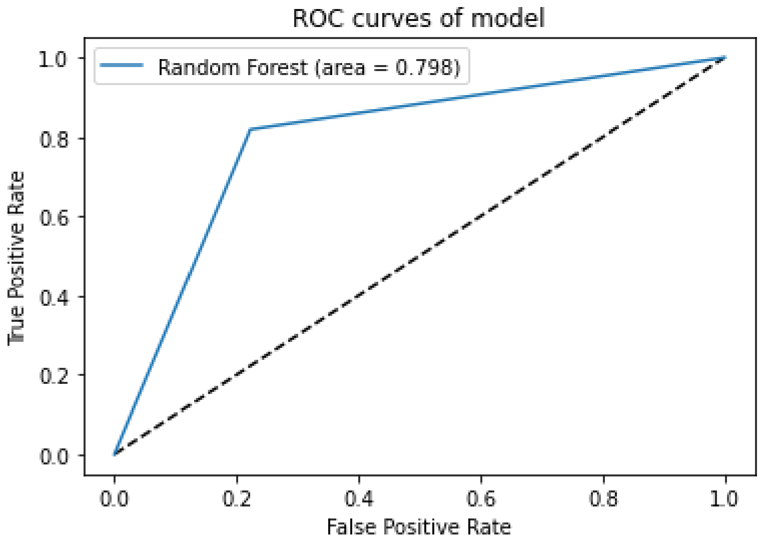
Out[49]:

```
{'bootstrap': False, 'criterion': 'entropy', 'n_estimators': 300}
```

ROC Curve

In [50]:

```
from sklearn import metrics
from sklearn.metrics import (confusion_matrix, precision_recall_curve, auc, roc_curve, reca
                             classification_report, f1_score, average_precision_score, prec

# Random forest
fpr_RandomForestC, tpr_RandomForestC, thresholds_RandomForestC = roc_curve(y_test, predicti
roc_auc_RandomForestC = auc(fpr_RandomForestC, tpr_RandomForestC)
precision_RandomForestC, recall_RandomForestC, th_RandomForestC = precision_recall_curve(y_

plt.plot([0, 1], [0, 1], 'k--')
#Random Forest
plt.plot(fpr_RandomForestC, tpr_RandomForestC, label='Random Forest (area = %0.3f)' % roc_a
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curves of model')
plt.legend(loc='best')
plt.show()
```



SVC

In [51]:

```
from sklearn.svm import SVC

svc = SVC(random_state = 42)
svc.fit(X_train, y_train)
pred_svc = svc.predict(X_test)
print(classification_report(y_test, pred_svc))
print(confusion_matrix(y_test,pred_svc))
print("Accuracy for training: {}".format(svc.score(X_train, y_train)))
print("Accuracy for testing: {}\n\n".format(svc.score(X_test, y_test)))
```

```
              precision    recall  f1-score   support

           0       0.83      0.74      0.78        99
           1       0.61      0.73      0.66        55

    accuracy                           0.73       154
   macro avg       0.72      0.73      0.72       154
weighted avg       0.75      0.73      0.74       154

[[73 26]
 [15 40]]
Accuracy for training: 0.8503740648379052
Accuracy for testing: 0.7337662337662337
```

Logistic Regression

In [52]:

```
from sklearn.linear_model import LogisticRegression
lgs = LogisticRegression(random_state=42)
lgs.fit(X_train, y_train)
pred_lgs = lgs.predict(X_test)
print(classification_report(y_test, pred_lgs))
print(confusion_matrix(y_test,pred_lgs))
print("Accuracy for training: {}".format(lgs.score(X_train, y_train)))
print("Accuracy for testing: {}\n\n".format(lgs.score(X_test, y_test)))
```

```
              precision    recall  f1-score   support

           0       0.82      0.72      0.76        99
           1       0.58      0.71      0.64        55

    accuracy                           0.71       154
   macro avg       0.70      0.71      0.70       154
weighted avg       0.73      0.71      0.72       154

[[71 28]
 [16 39]]
Accuracy for training: 0.7543640897755611
Accuracy for testing: 0.7142857142857143
```

In [53]:

```
parameters = {
     'penalty' : ['l1', 'l2'],
    'C' : np.logspace(-4, 4, 20),
    'solver' : ['liblinear', 'sag', 'newton-cg'],
    'class_weight':['dict', 'balanced']
    }
gcv = GridSearchCV(
    estimator = lgs ,
    param_grid = parameters,
    scoring = 'accuracy',
    cv = 10,
    n_jobs = -1
    )

gcv = gcv.fit(X_train, y_train)
gcv.best_score_
```

Out[53]:

```
0.7531172839506173
```

Neural Network

In [54]:

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(max_iter=1000, random_state=42, hidden_layer_sizes=(15,10,))
mlp.fit(X_train, y_train)
pred_mlp = mlp.predict(X_test)
print(classification_report(y_test, pred_mlp))
print(confusion_matrix(y_test,pred_mlp))
print("Accuracy for training: {}".format(mlp.score(X_train, y_train)))
print("Accuracy for testing: {}\n\n".format(mlp.score(X_test, y_test)))
```

```
              precision    recall  f1-score   support

           0       0.85      0.73      0.78        99
           1       0.61      0.76      0.68        55

    accuracy                           0.74       154
   macro avg       0.73      0.75      0.73       154
weighted avg       0.76      0.74      0.75       154

[[72 27]
 [13 42]]
Accuracy for training: 0.8241895261845387
Accuracy for testing: 0.7402597402597403
```

KNN

In [55]:

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
pred_knn = knn.predict(X_test)
print(classification_report(y_test, pred_knn))
print(confusion_matrix(y_test,pred_knn))
print("Accuracy for training: {}".format(knn.score(X_train, y_train)))
print("Accuracy for testing: {}\n\n".format(knn.score(X_test, y_test)))
```

```
              precision    recall  f1-score   support

           0       0.85      0.67      0.75        99
           1       0.57      0.78      0.66        55

    accuracy                           0.71       154
   macro avg       0.71      0.72      0.70       154
weighted avg       0.75      0.71      0.71       154

[[66 33]
 [12 43]]
Accuracy for training: 0.8615960099750624
Accuracy for testing: 0.7077922077922078
```

Decision Tree

In [56]:

```
from sklearn.tree import DecisionTreeClassifier
modelTree=DecisionTreeClassifier(random_state=42 , criterion = 'entropy', max_leaf_nodes =
                                 splitter= 'best')
modelTree.fit(X_train,y_train)
pred_Tree = modelTree.predict(X_test)
print(classification_report(y_test, pred_Tree))
print(confusion_matrix(y_test, pred_Tree))
print("Accuracy for training: {}".format(modelTree.score(X_train, y_train)))
print("Accuracy for testing: {}\n\n".format(modelTree.score(X_test, y_test)))
```

```
              precision    recall  f1-score   support

           0       0.84      0.75      0.79        99
           1       0.62      0.75      0.68        55

    accuracy                           0.75       154
   macro avg       0.73      0.75      0.73       154
weighted avg       0.76      0.75      0.75       154

[[74 25]
 [14 41]]
Accuracy for training: 0.9077306733167082
Accuracy for testing: 0.7467532467532467
```

In [57]:

```
parameters = {
     'criterion': ['gini', 'entropy'],
    'splitter' : ['best', 'random'],
    'max_leaf_nodes': list(range(2, 100)),
    'min_samples_split': [2, 3, 4]
    }
gcv = GridSearchCV(
    estimator = modelTree ,
    param_grid = parameters,
    scoring = 'accuracy',
    cv = 10,
    n_jobs = -1
    )

gcv = gcv.fit(X_train, y_train)
gcv.best_score_
```

Out[57]:

```
0.801820987654321
```

In [58]:

```
gcv.best_params_
```

Out[58]:

```
{'criterion': 'entropy',
 'max_leaf_nodes': 46,
 'min_samples_split': 3,
 'splitter': 'best'}
```

Naive Baiyes

In [59]:

```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(X_train,y_train)
pred_nb = nb.predict(X_test)
print(classification_report(y_test, pred_nb))
print(confusion_matrix(y_test,pred_nb))
print("Accuracy for training: {}".format(nb.score(X_train, y_train)))
print("Accuracy for testing: {}\n\n".format(nb.score(X_test, y_test)))
```

```
              precision    recall  f1-score   support

           0       0.80      0.74      0.77        99
           1       0.59      0.67      0.63        55

    accuracy                           0.71       154
   macro avg       0.69      0.71      0.70       154
weighted avg       0.73      0.71      0.72       154

[[73 26]
 [18 37]]
Accuracy for training: 0.7356608478802993
Accuracy for testing: 0.7142857142857143
```

XGBOOST

In [60]:

```
from xgboost import XGBClassifier

xgbc = XGBClassifier(random_state = 42)
xgbc.fit(X_train, y_train)
pred_xgbc = xgbc.predict(X_test)
print(classification_report(y_test, pred_xgbc))
print(confusion_matrix(y_test,pred_xgbc))
print("Accuracy for training: {}".format(xgbc.score(X_train, y_train)))
print("Accuracy for testing: {}\n\n".format(xgbc.score(X_test, y_test)))
```

```
              precision    recall  f1-score   support

           0       0.81      0.67      0.73        99
           1       0.55      0.73      0.62        55

    accuracy                           0.69       154
   macro avg       0.68      0.70      0.68       154
weighted avg       0.72      0.69      0.69       154

[[66 33]
 [15 40]]
Accuracy for training: 1.0
Accuracy for testing: 0.6883116883116883
```

5. Evaluation

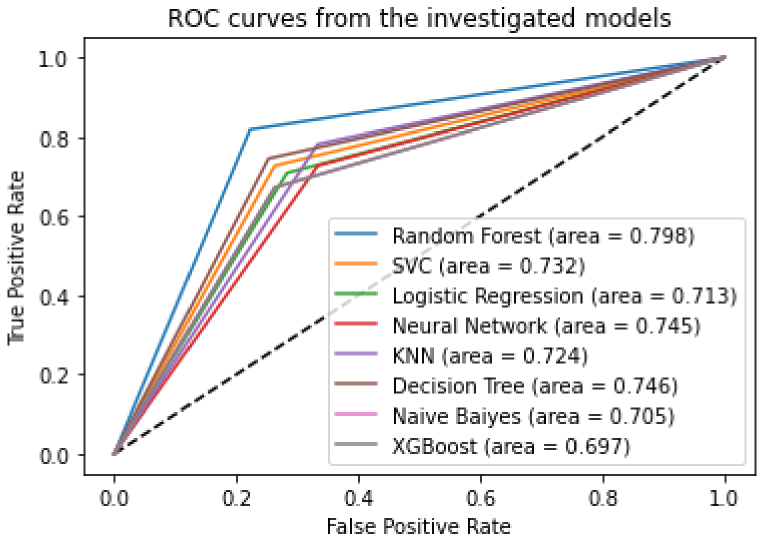
5.1 ROC Curve from the investigated models

In [61]:

```
#SVC
fpr_svc, tpr_svc, thresholds_svc = roc_curve(y_test, pred_svc)
roc_auc_svc = auc(fpr_svc, tpr_svc)
precision_svc, recall_svc, th_svc = precision_recall_curve(y_test, pred_svc)

# Logistic regression
fpr_lgs, tpr_lgs, thresholds_lgs = roc_curve(y_test, pred_lgs)
roc_auc_lgs = auc(fpr_lgs, tpr_lgs)
precision_lgs, recall_lgs, th_lgs = precision_recall_curve(y_test, pred_lgs)

#KNN
fpr_knn, tpr_knn, thresholds_knn = roc_curve(y_test, pred_knn)
roc_auc_knn = auc(fpr_knn, tpr_knn)
precision_knn, recall_knn, th_knn = precision_recall_curve(y_test, pred_knn)

#Decision Tree
fpr_Tree, tpr_Tree, thresholds_Tree = roc_curve(y_test, pred_Tree)
roc_auc_Tree = auc(fpr_Tree, tpr_Tree)
precision_Tree, recall_Tree, th_Tree = precision_recall_curve(y_test, pred_Tree)

#Naive Baiyes
fpr_nb, tpr_nb, thresholds_nb = roc_curve(y_test, pred_nb)
roc_auc_nb = auc(fpr_nb, tpr_nb)
precision_nb, recall_nb, th_nb = precision_recall_curve(y_test, pred_nb)

#Neural Network
fpr_mlp, tpr_mlp, thresholds_mlp = roc_curve(y_test, pred_mlp)
roc_auc_mlp = auc(fpr_mlp, tpr_mlp)
precision_mlp, recall_mlp, th_mlp = precision_recall_curve(y_test, pred_mlp)

#XGBoost
fpr_mlp, tpr_mlp, thresholds_mlp = roc_curve(y_test, pred_xgbc)
roc_auc_xgbc = auc(fpr_mlp, tpr_mlp)
precision_mlp, recall_mlp, th_mlp = precision_recall_curve(y_test, pred_xgbc)
```

In [62]:

```
plt.plot([0, 1], [0, 1], 'k--')
#Random Forest
plt.plot(fpr_RandomForestC, tpr_RandomForestC, label='Random Forest (area = %0.3f)' % roc_a
#SVC
plt.plot(fpr_svc, tpr_svc, label='SVC (area = %0.3f)' % roc_auc_svc)
#Logistic regression
plt.plot(fpr_lgs, tpr_lgs, label='Logistic Regression (area = %0.3f)' % roc_auc_lgs)
#Neural Network
plt.plot(fpr_mlp, tpr_mlp, label='Neural Network (area = %0.3f)' % roc_auc_mlp)
# KNN
plt.plot(fpr_knn, tpr_knn, label='KNN (area = %0.3f)' % roc_auc_knn)
#Decision Tree
plt.plot(fpr_Tree, tpr_Tree, label='Decision Tree (area = %0.3f)' % roc_auc_Tree)
#Naive Baiyes
plt.plot(fpr_nb, tpr_nb, label='Naive Baiyes (area = %0.3f)' % roc_auc_nb)
#Naive Baiyes
plt.plot(fpr_nb, tpr_nb, label='XGBoost (area = %0.3f)' % roc_auc_xgbc)


plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curves from the investigated models')
plt.legend(loc='best')
plt.show()
```



If the AUC is near to 1 which means it has good measure of separability. If the AUC is near to 0.5 which means model does not has good measure of separability.From the ROC Curve above, we can clearly see that the machine learning model that used Random Forest Classifier as classification algorithm gives us the highest AUC which is 0.798. It is means that Random Forest is able to distinguish between positive class and negative class. Therefore, we have choosen this classification algorithm to be appllied in our machine learning model.

Furthermore, the AUC of Neural Network and Decision Tree is very close. The AUC for Neural Network is 0.745 and Decision Tree is 0.746. The difference between them only 0.001. In addition, the AUC of model XGBoost is lower than other model, it only has 0.697. The AUC of XGBoost is near to 0.5, which means it does not has a good measure of separability.

5.2 Evaluate Random Forest Model

First Random Forest Model

In [63]:

```
print(classification_report(y_test, predictions))
print(confusion_matrix(y_test,predictions))
print("Accuracy for training: {}".format(RandomForestC.score(X_train, y_train)))
print("Accuracy for testing: {}\n\n".format(RandomForestC.score(X_test, y_test)))
```

```
              precision    recall  f1-score   support

           0       0.89      0.78      0.83        99
           1       0.67      0.82      0.74        55

    accuracy                           0.79       154
   macro avg       0.78      0.80      0.78       154
weighted avg       0.81      0.79      0.80       154

[[77 22]
 [10 45]]
Accuracy for training: 1.0
Accuracy for testing: 0.7922077922077922
```

Second Random Forest Model (Dropped Feature Importance Columns < 10%)

In [64]:

```
print(classification_report(y_test_drop10, predictions2))
print(confusion_matrix(y_test_drop10,predictions))
print("Accuracy for training: {}".format(RandomForestC2.score(X_train_drop10, y_train_drop1
print("Accuracy for testing: {}\n\n".format(RandomForestC2.score(X_test_drop10, y_test_drop
```

```
              precision    recall  f1-score   support

           0       0.83      0.76      0.79        99
           1       0.62      0.73      0.67        55

    accuracy                           0.75       154
   macro avg       0.73      0.74      0.73       154
weighted avg       0.76      0.75      0.75       154

[[77 22]
 [10 45]]
Accuracy for training: 1.0
Accuracy for testing: 0.7467532467532467
```

We have prepared 2 random forest model for evaluation. The first model is with complete dataset features where the second model has dropped the columns that have feature importance lower than 10%.

The first model have the accuracy of testing up to about 80% which is higher 5% than the second model. The first model gives recall up to 82% which shows it is able to predict the outcome with lesser false negative. False negative is contributes to the type II error which is unwanted in the medicine industry. As we stated in objectives, we want create a machine learning model with high accuracy and at the same time with less type II error occurs.

The second model is when we have dropped the columns such as the "Pregnancies", "SkinThickness" and "BloodPressure". After that, based on the results of the second model the overall accuracy dropped such as the precision and recall accuracy for both class(1 and 0). It means that the variable such as "pregnancies", "skinthickness" and "blood pressure" are contributing significantly for the random forest classifier model to do the diabetes prediction even though they only have less than 10% of feature importantance.

In short, we will choose the first random forest model as our machine learning model to predict the diabetes.

6. Deployment

Based on the evaluation result, we have selected random forest classifier model which takes all the features as input to do the diabetes prediction. As in medical field of point of view, the false negative rate of a machine learning model is to always be emphasized in minimizing it as much as possible because if a patient is predicted healthy, he or she will not care anymore and leave the hospital when actually he or she got the disease. In addition to that, the disease might already dragged for a long time for not to be cured and causes the patient to get into bad health condition or even worse such as death. Therefore, based on the recall rate, we have decided that the first random forest classifier model is the most suitable model to be deployed for the medical industry to implement it.

The api that we are going to use to deploy our machine learning model is Flask. Flask is a web framework which basically written in Python. Furthermore, it comes with a light-weighted server which has no databases and the configuration is just minimal. By using flask, we will create an url which specifically link to our machine learning model api. Besides that, we will also use the Postman api to test our machine learning model api to ensure it is working properly such as giving a series of input to our machine learning model api, we will get our expected output such as binary class output before we deploy this api to the real market. If we get what we wanted, then we should be happy because we have just created an api for the medical experts to use.

7. Conclusion

This study is mainly to predict the risk of diabetes. In this study, 8 Model were tested which are Random Forest, SVC, Logistic Regression, Neural Network, K Nearest Neighbor, Decision tree, Naive Baiyes and XGBoost. The accuracy of all model are between 60% to 80%.

Random Forest is considered the best in comparing with other models. It is because the accuracy of Random Forest is the highest that is around 80%. It can be concluded that Random Forest with feature selection achieved increased the classification performance.

As a conclusion, the model has achieved our goal which is perform well in diabetes prediction even though precision is low but the recall is maintained at 82%. Therefore, we would like to state the limitations and the future improvements that can be made to improve our project.

One of the limitation of this project is that we found the data quality is not good in terms of dataset completion. As we know, there are around 290 rows having missing values for that particular column which is "Insulin" for the training datset. Furthermore, there are also some other column having a number of missing values as well. We take "Insulin" column into account, if we remove the rows which has null value in that column, we would be losing approximately 38% of data from the dataset. Therefore, the dataset is not good enough. The second limitaiton would be the machine learning model is hardly to do the diabetes prediction which the best model only gives us 79% overall accuracy and precision accuracy for positive class is also very low which is 67%. It is most probably due to large number of missing values and even though we have impute it withe median of that parcticular column. Besides that, it does not seems to able to predict more accurately with the imputed dataset but the method that gives us the highest accuracy is by imputing dataset with median instead of mean or removing the missing value rows.

Lastly, one of the improvements to be made in the future is that we can try to use deep learning algorithm to build a machine learning model as the Random Forest Classifier and Logistic Regression are unable to achieve better accuracy such as precision and recall. Hence, we can try to do deep learning on diabetes prediction and see how good it is. Before that, we would also like to have the chance to get the complete dataset about diabetic patient and healthy patient from government hospital so that we can deal with real data and the dataset would be more complete.

8. Reference

Data source: Kaggle.com. 2020. Pima Indians Diabetes Database. [online] Available at: https://www.kaggle.com/uciml/pima-indians-diabetes-database (https://www.kaggle.com/uciml/pima-indians-diabetes-database).

CodeBlue. (2019). Diabetes, Hypertension Treatment Cost Government RM3Bil In 2019. [online] Available at: https://codeblue.galencentre.org/2019/12/24/diabetes-hypertension-treatment-cost-government-rm3bil-in-2019/ (https://codeblue.galencentre.org/2019/12/24/diabetes-hypertension-treatment-cost-government-rm3bil-in-2019/).

Medium. 2020. Understanding AUC - ROC Curve. [online] Available at: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5 (https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5) [Accessed 11 September 2020].

die, X., 2020. Read Random Forest-Random Forest (4 Implementation Steps + 10 Advantages And Disadvantages). [online] 产品经理的人工智能学习库. Available at: https://easyai.tech/en/ai-definition/random-forest/#yqd (https://easyai.tech/en/ai-definition/random-forest/#yqd) [Accessed 11 September 2020].

Kumar, N., Kumar, N. and profile, V., 2020. Advantages And Disadvantages Of Random Forest Algorithm In Machine Learning. [online] Theprofessionalspoint.blogspot.com. Available at: http://theprofessionalspoint.blogspot.com/2019/02/advantages-and-disadvantages-of-random.html (http://theprofessionalspoint.blogspot.com/2019/02/advantages-and-disadvantages-of-random.html) [Accessed 11 September 2020].

OpenGenus IQ: Learn Computer Science. 2020. Advantages And Disadvantages Of Logistic Regression. [online] Available at: https://iq.opengenus.org/advantages-and-disadvantages-of-logistic-regression/ (https://iq.opengenus.org/advantages-and-disadvantages-of-logistic-regression/) [Accessed 12 September 2020].

Linkedin.com. 2020. Artificial Neural Networks Advantages And Disadvantages. [online] Available at: https://www.linkedin.com/pulse/artificial-neural-networks-advantages-disadvantages-maad-m-mijwel (https://www.linkedin.com/pulse/artificial-neural-networks-advantages-disadvantages-maad-m-mijwel) [Accessed 12 September 2020].

Kumar, N., Kumar, N. and profile, V., 2020. Advantages And Disadvantages Of KNN Algorithm In Machine Learning. [online] Theprofessionalspoint.blogspot.com. Available at: http://theprofessionalspoint.blogspot.com/2019/02/advantages-and-disadvantages-of-knn.html (http://theprofessionalspoint.blogspot.com/2019/02/advantages-and-disadvantages-of-knn.html) [Accessed 12 September 2020].

Medium. 2020. Top 5 Advantages And Disadvantages Of Decision Tree Algorithm. [online] Available at: https://medium.com/@dhiraj8899/top-5-advantages-and-disadvantages-of-decision-tree-algorithm-428ebd199d9a (https://medium.com/@dhiraj8899/top-5-advantages-and-disadvantages-of-decision-tree-algorithm-428ebd199d9a) [Accessed 12 September 2020].

Kumar, N., Kumar, N. and profile, V., 2020. Advantages And Disadvantages Of Naive Bayes In Machine Learning. [online] Theprofessionalspoint.blogspot.com. Available at: http://theprofessionalspoint.blogspot.com/2019/03/advantages-and-disadvantages-of-naive.html (http://theprofessionalspoint.blogspot.com/2019/03/advantages-and-disadvantages-of-naive.html) [Accessed 12 September 2020].

Kumar, N., Kumar, N. and profile, V., 2020. Advantages Of Xgboost Algorithm In Machine Learning. [online] Theprofessionalspoint.blogspot.com. Available at: http://theprofessionalspoint.blogspot.com/2019/03/advantages-of-xgboost-algorithm-in.html (http://theprofessionalspoint.blogspot.com/2019/03/advantages-of-xgboost-algorithm-in.html) [Accessed 12 September 2020].