Diabetes prediction using different classification models, calculating accuracy of each model and finding the model which gives high accuracy

Diabetes is a type of chronic disease which is more common among the people of all age groups. Predicting this disease at an early stage can help a person to take the necessary precautions and change his/her lifestyle accordingly to either prevent the occurrence of this disease or control the disease(For people who already have the disease).

Steps:
--> Loading Dataset
--> Visualizing and cleaning Dataset
--> Spliting the dataset into training set and testing set
--> Building differnt classification models, predicting values and accuracy
--> Finding the highest accuracy

LOADING DATASET

Importing libraries pandas, numpy and matplotlib

In [14]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Reading data set , these dataset is regarding the features of diffent people and outcome int the form of binary number 1 or 0. 1 - indicates diabetics is present, 0- indicates diabetics is absent. This dataset is downloaded from UCI machine learning repository

In [15]:
```
df = pd.read_csv(r'Documents\Machine-Learning-with-Python-master\diabete
s.csv')
```

Analyzing and Visualizing Dataset

In [4]:
```
df.head()
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	6	148	72	35	0	33.6	0.627
1	1	85	66	29	0	26.6	0.351
2	8	183	64	0	0	23.3	0.672
3	1	89	66	23	94	28.1	0.167
4	0	137	40	35	168	43.1	2.288

We can observe from the above data set it has different features like Pregnancies, Glucose, BloodPressure, Skin Thickness, Insulin , BMI , DiabetesPedigreeFunction, Age. Here label is Outcome

In [5]:
```
print("shape of data set is {}".format(df.shape))
```

```
shape of data set is (768, 9)
```

The above data set has total 768 rows and 9 columns

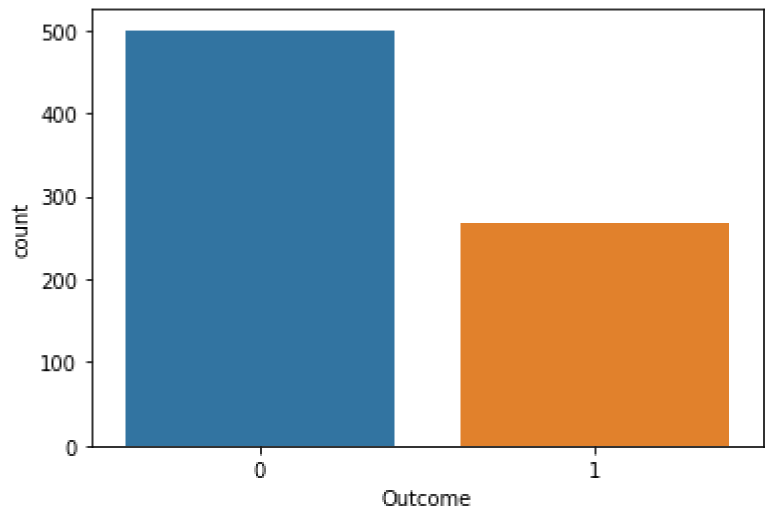
In [6]:
```
df.groupby('Outcome').size()
```

Out[6]:
```
Outcome
0    500
1    268
dtype: int64
```

It has total 268 Diabetes positive cases and 500 diabetes negative cases

In [26]:
```
import seaborn as sns
sns.countplot(df['Outcome'],label='count')
```

Out[26]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1f665eaf748>
```



STATISTICAL MEASUREMENTS

we can also calculate different statistical measurements like mean, standard deviation of each feature of a data set

In [5]:
```
df.describe()
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesP
count	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	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

We can get data types of differnt features by using the below the Data code

In [21]:
```
df.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
```

SPLITING THE DATASET INTO TRAINING AND TESTING SETS
TRAINING SET : The part of the dataset which we are going to train to the model
TESTING SET : The part of the dataset on which we are going to test out model
Inorder to split the data into training and testing data set we should import the train_test_split from the sklearn module

In [16]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(df.loc[:,df.columns!='O
utcome'],df['Outcome'],stratify=df['Outcome'],random_state=66)
```

Features which we are going to train

In [9]:
```
x_train.head()
```

Out[9]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
612	7	168	88	42	321	38.2	0.787
557	8	110	76	0	0	27.8	0.237
26	7	147	76	0	0	39.4	0.257
70	2	100	66	20	90	32.9	0.867
73	4	129	86	20	270	35.1	0.231

Features set of the testing data

```
In [8]: x_test.head()
```

Out[8]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
506	0	180	90	26	90	36.5	0.314
709	2	93	64	32	160	38.0	0.674
257	2	114	68	22	0	28.7	0.092
518	13	76	60	0	0	32.8	0.180
432	1	80	74	11	60	30.0	0.527

Label set of the Training part of the dataset

```
In [9]: y_train.head()
```

```
Out[9]: 612    1
557    0
26     1
70     1
73     0
Name: Outcome, dtype: int64
```

Label set of the Testing part of the dataset

```
In [10]: y_test.head()
```

```
Out[10]: 506    1
709    1
257    0
518    0
432    0
Name: Outcome, dtype: int64
```

In this project the classification models used are:
1) k Nearest neighbors model
2) Logistic regression model

3) Decision Trees
4) Support Vector Machine

Building model using k-nearest neighbors classification

The k-NN algorithm is the simplest machine learning algorithm. Building the model consists only of storing the training data set. To make a prediction for a new data point, the algorithm finds the closest data points in the training data set — its “nearest neighbors.”

Let us predict the outcome with differnt values of k (no_of_neighbors)

In [17]:
```
# K-nearest_neighbors classifer
list_of_training_accuracy = []
list_of_testing_accuracy = []
from sklearn.neighbors import KNeighborsClassifier
training_accuracy = []
testing_accuracy = []
neighbors = list(range(1,10))
for no_of_neighbors in neighbors:
    kn = KNeighborsClassifier(n_neighbors=no_of_neighbors).fit(x_train,y
_train)
    y_pre = kn.predict(x_test)
    print("predicted values of k = {} is ".format(no_of_neighbors))
    print(y_pre)
```

```
predicted values of k = 1 is
[1 0 0 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0 1 1 0 0 1 0 0 0 0 1 1 1 1 1 0 0 0 0
0
 0 0 1 1 0 1 0 1 0 0 1 0 1 0 1 0 0 1 1 1 1 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0
0
 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 1 1 1 0 1 1 1 0 0 1
1
 0 1 0 0 0 0 1 0 1 1 0 0 1 1 0 1 0 1 0 0 0 1 0 0 0 1 1 0 0 1 1 1 1 1 0 0
0
 1 1 1 0 1 0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 1 1 0 1 1 0 0 1 0 0 1 1 0 0 1 1
0
 0 0 1 0 0 0 0]
predicted values of k = 2 is
[1 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
 0 0 0 1 0 0 0 1 0 0 1 0 1 0 1 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 1 0 0 0 1 0 0 1 0 1 0 1 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0
1
 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 1 0 0
0
 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1
0
 0 0 1 0 0 0 0]
predicted values of k = 3 is
[1 0 0 0 0 1 1 0 0 1 1 0 1 0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 0 0 1 1 0 0 0 1
0
 0 1 0 1 0 0 0 1 0 0 1 0 1 0 1 0 0 1 1 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0

 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 0 1 1 1 0 0 1 1 1 0 0
1
 0 1 0 0 1 0 1 0 1 0 0 0 1 1 1 0 1 0 0 0 0 1 0 0 0 1 1 0 0 1 1 1 1 0 0
0
 0 1 1 0 1 1 0 1 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 1 0 1 0 1
0
 1 0 1 0 1 0 0]
predicted values of k = 4 is
[1 0 0 0 0 1 1 0 0 1 1 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
 0 1 0 1 0 0 0 1 0 0 1 0 1 0 1 0 0 1 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0
0
 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 0 0 0
0
 0 1 0 0 0 0 1 0 0 0 0 0 0 1 1 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 0 0
0
 0 1 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1
0
 1 0 1 0 1 0 0]
predicted values of k = 5 is
[1 0 0 0 0 1 1 1 0 1 1 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0
1
 1 1 0 1 0 0 0 1 0 0 1 0 1 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0
0
 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 1 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 0 0 0
1
 0 1 0 0 0 0 1 0 0 0 0 0 0 1 1 1 0 1 0 0 0 0 1 0 0 0 1 1 0 0 1 1 1 1 0 0
0
 1 1 0 0 1 0 0 1 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1
0
 1 0 1 0 1 1 0]
predicted values of k = 6 is
[0 0 0 0 0 1 1 0 0 0 1 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
[0 0 0 0 0 1 1 0 0 0 1 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
 0 1 0 1 0 0 0 1 0 0 1 0 1 0 1 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0
0
 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 0 0 0
1
 0 1 0 0 0 0 1 0 0 0 0 0 0 1 1 1 0 1 0 0 0 0 1 0 0 0 1 1 0 0 0 1 1 1 0 0
0
 0 1 0 0 1 0 0 1 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1
0
 1 0 1 0 1 1 0]
predicted values of k = 7 is
[1 0 0 0 0 1 1 0 0 0 1 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0

 0 1 0 1 0 0 0 1 0 0 1 0 1 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0
0
 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 1 1 0 0 1 1 0 0 0 0 1 1 1 0 0 0 1 1 0 0
1
 1 1 0 0 0 0 1 0 0 0 0 0 0 1 1 1 0 1 0 0 0 0 1 0 0 1 1 1 0 0 0 1 1 1 0 0
1
 1 1 0 0 1 0 0 1 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 1 1
1
 1 0 1 1 1 1 0]
predicted values of k = 8 is
[1 0 0 0 0 1 1 0 0 0 1 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0
 0 1 0 1 0 0 0 1 0 0 1 0 1 0 1 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0
0
 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 1 0 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0
0
 1 1 0 0 0 0 1 0 0 0 0 0 0 1 1 1 0 1 0 0 0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0
0
 1 1 0 0 1 0 0 1 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1
0
 0 0 1 0 1 0 0]
predicted values of k = 9 is
[1 0 0 0 0 1 1 0 0 0 1 0 1 0 0 0 0 1 1 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0
 1 1 0 1 0 0 0 1 0 0 1 0 1 0 1 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0
0
 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 1 1 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0
1
 1 1 0 0 1 0 1 0 0 0 0 0 0 1 1 1 0 1 0 0 1 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0
1
 1 1 0 0 1 0 0 1 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 1 1
0
 0 0 1 0 1 0 0]
 0 0 1 0 1 0 0]
```

Let us calculate the accuracy of each model for each value of K which are used above

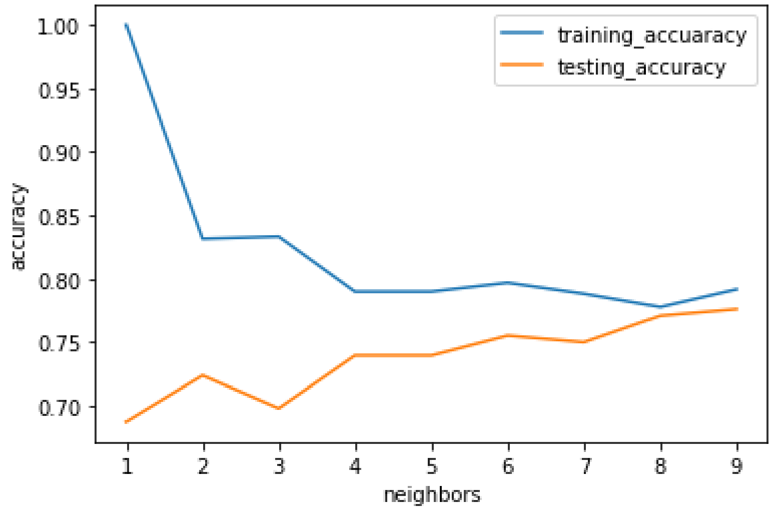
```
In [18]: from sklearn.metrics import confusion_matrix
         from sklearn.metrics import classification_report
         for no_of_neighbors in neighbors:
             kn = KNeighborsClassifier(n_neighbors=no_of_neighbors).fit(x_train,y
         _train)
             training_accuracy.append(kn.score(x_train,y_train))
             testing_accuracy.append(kn.score(x_test,y_test))
         print("list of training accuracy of differnt k values models")
         print(training_accuracy)
         print('\n')
         print('list of testing accuracy of differnt k values models')
         print(testing_accuracy)
```

```
list of training accuracy of differnt k values models
[1.0, 0.8315972222222222, 0.8333333333333334, 0.7899305555555556, 0.7899
305555555556, 0.796875, 0.7881944444444444, 0.7777777777777778, 0.791666
6666666666]


list of testing accuracy of differnt k values models
[0.6875, 0.7239583333333334, 0.6979166666666666, 0.7395833333333334, 0.7
395833333333334, 0.7552083333333334, 0.75, 0.7708333333333334, 0.7760416
666666666]
```

Plotting the graph of between accuracies of training and testing datasets for differnt values of k

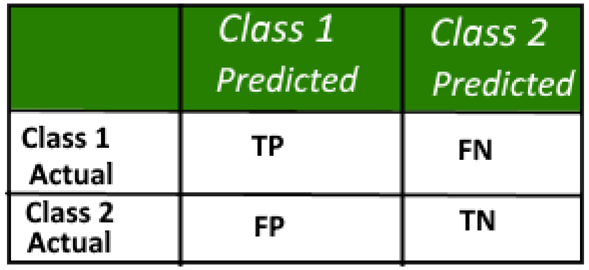
```
In [19]: plt.plot(neighbors,training_accuracy,label="training_accuaracy")
         plt.plot(neighbors,testing_accuracy,label="testing_accuracy")
         plt.ylabel('accuracy')
         plt.xlabel('neighbors')
         plt.legend()
         plt.savefig('knn_accuracy_comapare_model')
```



from the above graph we can observe that we should we are accuracy of training and testing dataset at equal level at around 8-9, hence we can consider that k value around 9

Hence for k equal to around 8-9 we are getting the matching accuracy for both training and testing sets. The model is built with the value of k = 9

Confusion Matrix: A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix. The confusion matrix shows the ways in which your classification model is confused when it makes predictions. It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that are being made.



	Class 1 Predicted	Class 2 Predicted
Class 1 Actual	TP	FN
Class 2 Actual	FP	TN

Here, Class 1 : Positive Class 2 : Negative

Definition of the Terms:

Positive (P) : Observation is positive (for example: is an apple). Negative (N) : Observation is not positive (for example: is not an apple). True Positive (TP) : Observation is positive, and is predicted to be positive. False Negative (FN) : Observation is positive, but is predicted negative. True Negative (TN) : Observation is negative, and is predicted to be negative. False Positive (FP) : Observation is negative, but is predicted positive

$$\textbf{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\textbf{\textit{Recall}} = \frac{TP}{TP + FN}$$

$$\textbf{\textit{Precision}} = \frac{TP}{TP + FP}$$

High recall, low precision: This means that most of the positive examples are correctly recognized (low FN) but there are a lot of false positives.

Low recall, high precision: This shows that we miss a lot of positive examples (high FN) but those we predict as positive are indeed positive (low FP)

F-measure: Since we have two measures (Precision and Recall) it helps to have a measurement that represents both of them. We calculate an F-measure which uses Harmonic Mean in place of Arithmetic Mean as it punishes the extreme values more. The F-Measure will always be nearer to the smaller value of Precision or Recall.

$$F\text{ - measure} = \frac{2*\text{Recall}*\text{Precision}}{\text{Recall} + \text{Precision}}$$

In [20]:
```
# from the above figure we can say that we have to neighbor some where a
round 9
kn = KNeighborsClassifier(n_neighbors = 9).fit(x_train,y_train)
list_of_training_accuracy.append(kn.score(x_train,y_train))
list_of_testing_accuracy.append(kn.score(x_test,y_test))
print("Accuracy of the K-Neares_neighbors_classifer on training set is "
,kn.score(x_train,y_train))
print("Accuracy of the K-Neares_neighbors_classifer on testing set is ",
kn.score(x_test,y_test))
print()
print("Confusion Matrix ")
print(confusion_matrix(y_test,kn.predict(x_test)))
print()
print("Report")
print(classification_report(y_test,kn.predict(x_test)))
```

```
Accuracy of the K-Neares_neighbors_classifer on training set is  0.79166
66666666666
Accuracy of the K-Neares_neighbors_classifer on testing set is  0.776041
6666666666

Confusion Matrix
[[105  20]
 [ 23  44]]
```

```
Report
              precision    recall  f1-score   support

           0       0.82      0.84      0.83       125
           1       0.69      0.66      0.67        67

    accuracy                           0.78       192
   macro avg       0.75      0.75      0.75       192
weighted avg       0.77      0.78      0.77       192
```

Building model using logistic regression

This type of statistical analysis (also known as logit model) is often used for predictive analytics and modeling, and extends to applications in machine learning. In this analytics approach, the dependent variable is finite or categorical: either A or B (binary regression) or a range of finite options A, B, C or D (multinomial regression). It is used in statistical software to understand the relationship between the dependent variable and one or more independent variables by estimating probabilities using a logistic regression equation.

This type of analysis can help you predict the likelihood of an event happening or a choice being made. For example, you may want to know the likelihood of a visitor choosing an offer made on your website — or not (dependent variable). Your analysis can look at known characteristics of visitors, such as sites they came from, repeat visits to your site, behavior on your site (independent variables). Logistic regression models help you determine a probability of what type of visitors are likely to accept the offer — or not. As a result, you can make better decisions about promoting your offer or make decisions about the offer itself.

In [21]:
```python
# logistic Regression
from sklearn.linear_model import LogisticRegression
lgre = LogisticRegression().fit(x_train,y_train)
ypre = lgre.predict(x_test)
print("predicted values",ypre)
print("test set values",list(y_test))
list_of_training_accuracy.append(lgre.score(x_train,y_train))
list_of_testing_accuracy.append(lgre.score(x_test,y_test))
print()
print("Accuracy of Logistic_Regression on training set is ",lgre.score(x
_train,y_train))
print("Accuracy of Logistic_Regression on testing set is ",lgre.score(x_
test,y_test))
```

```
print()
print("Confusion Matrix ")
print(confusion_matrix(y_test,lgre.predict(x_test)))
print()
print("Report")
print(classification_report(y_test,lgre.predict(x_test)))
```

```
C:\anaconda\lib\site-packages\sklearn\linear_model\logistic.py:432: Futu
reWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
```

```
predicted values [1 0 0 0 0 1 1 0 0 0 1 0 1 0 1 0 0 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
 1 1 0 1 0 0 0 1 0 0 1 0 1 1 1 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 1 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0
1
 0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 1 0 1 0 0 1 0 0 0 0 0 0 1 0 0 1 1 1 1 0 0
1

 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 1
1
 0 0 0 0 1 0 0]
test set values [1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1,
1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0,
0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0,
0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1,
1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0,
0, 1, 0, 1, 1, 0]

Accuracy of Logistic_Regression on training set is  0.78125
Accuracy of Logistic_Regression on testing set is  0.7708333333333334

Confusion Matrix
[[110  15]
 [ 29  38]]

Report
              precision    recall  f1-score   support

           0       0.79      0.88      0.83       125
           1       0.72      0.57      0.63        67
```

```
    accuracy                           0.77       192
   macro avg       0.75      0.72      0.73       192
weighted avg       0.77      0.77      0.76       192
```

Building a model using Decision tree

Decision tree algorithm falls under the category of supervised learning. They can be used to solve both regression and classification problems. Decision tree uses the tree representation to solve the problem in which each leaf node corresponds to a class label and attributes are represented on the internal node of the tree. We can represent any boolean function on discrete attributes using the decision tree.

In [22]:
```
#decision trees

from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(max_depth = 3).fit(x_train,y_train)
ypre=dtree.predict(x_test)
print("Predicted values of Outcome using Decison Trees classfier")
print(ypre)
print()
print("actual values of Outcome of the order set")
print(list(y_test))
list_of_training_accuracy.append(dtree.score(x_train,y_train))
list_of_testing_accuracy.append(dtree.score(x_test,y_test))
print()
print("Accuracy of Decision Trees on training set is ",dtree.score(x_tra
in,y_train))
print("Accuracy of Decision Trees on testing set is ",dtree.score(x_test
,y_test))
print()
print("Confusion Matrix ")
print(confusion_matrix(y_test,kn.predict(x_test)))
print()
print("Report")
print(classification_report(y_test,kn.predict(x_test)))
```

```
Predicted values of Outcome using Decison Trees classfier
[1 0 0 0 0 1 1 0 0 0 1 0 1 0 1 0 0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 1
1
 1 0 0 1 0 0 0 1 0 0 1 0 1 1 1 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0
0
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 1 0 0 0 0 0 1 0 1 0 1 0 0 1 0 0
1
1
 0 1 0 0 1 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 1 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0
1
 0 0 0 0 1 1 0 1 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 1 0 0 1 1
0
 0 0 0 0 1 0 0]

actual values of Outcome of the order set
[1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1,
0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1,
1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0,
0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1,
0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0]

Accuracy of Decision Trees on training set is  0.7725694444444444
Accuracy of Decision Trees on testing set is  0.7395833333333334

Confusion Matrix
[[105  20]
 [ 23  44]]


Report
              precision    recall  f1-score   support

           0       0.82      0.84      0.83       125
           1       0.69      0.66      0.67        67

    accuracy                           0.78       192
   macro avg       0.75      0.75      0.75       192
weighted avg       0.77      0.78      0.77       192
```

Building a model using Support Vector Machine

SVM is a supervised machine learning algorithm which can be used for classification or regression problems. It uses a technique called the kernel trick to transform your data and then based on these transformations it finds an optimal boundary between the possible outputs. Simply put, it does some extremely complex data transformations, then figures out how to seperate your data based on the labels or outputs you've defined.

```
In [23]: #support vector Machine
         from sklearn.svm import SVC
         svc = SVC()
         svc.fit(x_train, y_train)
         print("Accuracy on training set: {:.2f}".format(svc.score(x_train, y_tra
         in)))
         print("Accuracy on test set: {:.2f}".format(svc.score(x_test, y_test)))
         print()
         print("Confusion Matrix ")
         print(confusion_matrix(y_test,kn.predict(x_test)))
         print()
         print("Report")
         print(classification_report(y_test,kn.predict(x_test)))
```

```
Accuracy on training set: 1.00
Accuracy on test set: 0.65

Confusion Matrix
[[105  20]
 [ 23  44]]

Report
              precision    recall  f1-score   support

           0       0.82      0.84      0.83       125
           1       0.69      0.66      0.67        67

    accuracy                           0.78       192
   macro avg       0.75      0.75      0.75       192
weighted avg       0.77      0.78      0.77       192
```

```
C:\anaconda\lib\site-packages\sklearn\svm\base.py:193: FutureWarning: Th
e default value of gamma will change from 'auto' to 'scale' in version
0.22 to account better for unscaled features. Set gamma explicitly to 'a
uto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

The accuracy on the training set is 100%, while the test set accuracy is much worse. This is an indicative that the tree is overfitting and not generalizing well to new data. Therefore, we need to apply pre-pruning to the tree. We set max_depth=3, limiting the depth of the tree decreases overfitting. This leads to a lower accuracy on the training set, but an improvement on the test set.

In [24]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.fit_transform(x_test)
svc = SVC()
svc.fit(x_train_scaled, y_train)
print("Accuracy on training set: {:.2f}".format(svc.score(x_train_scaled
, y_train)))
print("Accuracy on test set: {:.2f}".format(svc.score(x_test_scaled, y_t
est)))
print()
print("Confusion Matrix ")
print(confusion_matrix(y_test,kn.predict(x_test)))
print()
print("Report")
print(classification_report(y_test,kn.predict(x_test)))
```

```
Accuracy on training set: 0.77
Accuracy on test set: 0.77

Confusion Matrix
[[105  20]
 [ 23  44]]

Report
              precision    recall  f1-score   support

           0       0.82      0.84      0.83       125
           1       0.69      0.66      0.67        67

    accuracy                           0.78       192
   macro avg       0.75      0.75      0.75       192
weighted avg       0.77      0.78      0.77       192
```

```
C:\anaconda\lib\site-packages\sklearn\svm\base.py:193: FutureWarning: Th
e default value of gamma will change from 'auto' to 'scale' in version
0.22 to account better for unscaled features. Set gamma explicitly to 'a
uto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

In [25]:
```
svc = SVC(C=1000)
svc.fit(x_train_scaled, y_train)
list_of_training_accuracy.append(svc.score(x_train_scaled,y_train))
```

```
list_of_testing_accuracy.append(svc.score(x_test_scaled,y_test))
print("Accuracy on training set: {:.3f}".format(
    svc.score(x_train_scaled, y_train)))
print("Accuracy on test set: {:.3f}".format(svc.score(x_test_scaled, y_t
est)))
print()
print("Confusion Matrix ")
print(confusion_matrix(y_test,kn.predict(x_test)))
print()
print("Report")
print(classification_report(y_test,kn.predict(x_test)))
```

```
Accuracy on training set: 0.790
Accuracy on test set: 0.797


Confusion Matrix
[[105  20]
 [ 23  44]]

Report
              precision    recall  f1-score   support

           0       0.82      0.84      0.83       125
           1       0.69      0.66      0.67        67

    accuracy                           0.78       192
   macro avg       0.75      0.75      0.75       192
weighted avg       0.77      0.78      0.77       192

```

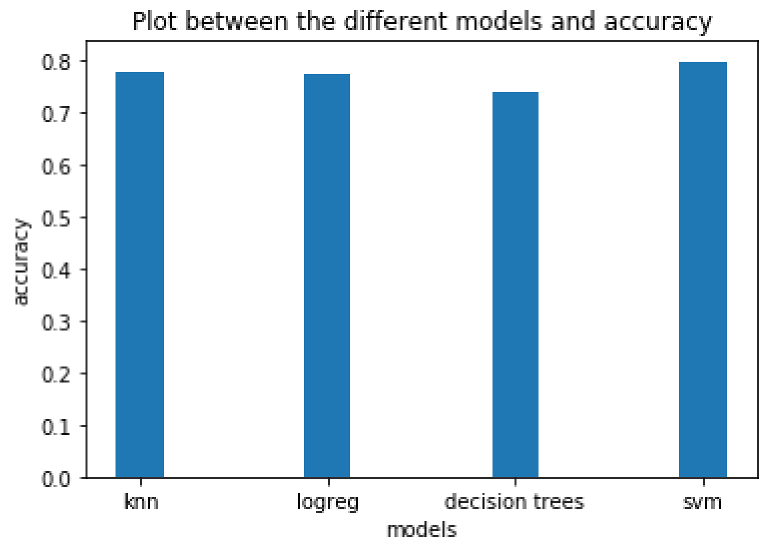
```
C:\anaconda\lib\site-packages\sklearn\svm\base.py:193: FutureWarning: Th
e default value of gamma will change from 'auto' to 'scale' in version
0.22 to account better for unscaled features. Set gamma explicitly to 'a
uto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

PLOTING BAR GRAPH BETWEEN DIFFERENT MODELS AND ACCURACY RESPECTIVELY

```
In [26]: list_of_models=['knn','logreg','decision trees','svm']
         plt.bar(list_of_models,list_of_testing_accuracy,width = 0.25)
         plt.xlabel('models')
         plt.ylabel('accuracy')
         plt.title('Plot between the different models and accuracy')
```

```
Out[26]: Text(0.5, 1.0, 'Plot between the different models and accuracy')
```

```
Out[26]: Text(0.5, 1.0, 'Plot between the different models and accuracy')
```



From the above graph we can observe that the almost all the models have accuracy closly, comparing to Support Vector Machine has the high accuracy comparing to all the models

```
In [32]: table = pd.DataFrame({'Model_name':list_of_models,'Accuracy':list_of_tes
ting_accuracy})
print(table)
```

```
       Model_name  Accuracy
0             knn  0.776042
1          logreg  0.770833
2  decision trees  0.739583
3             svm  0.796875
```

Conclusion: The diabetes data set is trained to differnt models ,built the diiferent models and calculated the accuracy of each model. Among all the models support vector machine has the highest accuracy.