DISEASE PREDICTION

A PROJECT REPORT

In partial fulfilment of the requirements for the award of the degree

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS & COMMUNICATION

Under the guidance of

SANDIPAN GUPTA

SUBMITTED BY

SOUMYAJIT KUNDU

SAPTARSHI SARKAR

SHRESTHA GHOSH

SHIS ROY CHOWDHURY

TIYASHA GUHA

ACADEMY OF TECHNOLOGY

In association with

(ISO9001:2015)

SDF Building, Module #132, Ground Floor, Salt Lake City, GP Block, Sector V, Kolkata, West Bengal 700091

(Note: All entries of the proforma of approval should be filled up with appropriate and complete information. Incomplete proforma of approval in any respect will be summarily rejected.)

1. Title of the Project: **DISEASE PREDICTION**
2. Project Members: **SOUMYAJIT KUNDU**
   SAPTARSHI SARKAR
   SHRESTHA GHOSH
   SHIS ROY CHOWDHURY
   TIYASHA GUHA

3. Name of the guide: **Mr. SANDIPAN GUPTA**
4. Address: Ardent Computech Pvt. Ltd
   (An ISO 9001:2015 Certified)
   SDF Building, Module #132, Ground Floor, Salt Lake City, GP Block, Sector V, Kolkata, West Bengal, 700091

Project Version Control History

Version	Primary Author	Description of Version	Date Completed
Final	Soumyajit Kundu Saptarshi Sarkar Shrestha Ghosh Shis Roy Chowdhury Tiyasha Guha	Project Report	11th July,2019

1.
2.
3.
4.
5.

Signature of Team Members

Date:

Signature of Approver

Date:

For Office Use Only

Approved

Not Approved

MR.SANDIPAN GUPTA

Project Proposal Evaluator

DECLARATION

We hereby declare that the project work being presented in the project proposal entitled **"DISEASE PREDICTION"** in partial fulfilment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY** at **ARDENT COMPUTECH PVT. LTD, SALTLAKE, KOLKATA, WEST BENGAL,** is an authentic work carried out under the guidance of **MR. SANDIPAN GUPTA**. The matter embodied in this project work has not been submitted elsewhere for the award of any degree of our knowledge and belief.

Date:

Name of the Student: Soumyajit Kundu

Saptarshi Sarkar

Shrestha Ghosh

Shis Roy Chowdhury

Tiyasha Guha

Signature of the students:

1.

2.

3.

4.

5.

Ardent Computech Pvt. Ltd (An ISO 9001:2015 Certified)

SDF Building, Module #132, Ground Floor, Salt Lake City, GP Block, Sector V, Kolkata, West Bengal 700091

CERTIFICATE

This is to certify that this proposal of minor project entitled **"DISEASE PREDICTION"** is a record of bonafide work, carried out by ***Soumyajit Kundu, Saptarshi Sarkar, Shrestha Ghosh, Shis Roy Chowdhury, Tiyasha Guha*** under my guidance at **ARDENT COMPUTECH PVT LTD**. In my opinion, the report in its present form is in partial fulfilment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY** and as per regulations of the **ARDENT®.** To the best of my knowledge, the results embodied in this report, are original in nature and worthy of incorporation in the present version of the report.

Guide / Supervisor

MR. SANDIPAN GUPTA

Project Engineer

Ardent Computech Pvt. Ltd (An ISO 9001:2015 Certified)

SDF Building, Module #132, Ground Floor, Salt Lake City, GP Block, Sector V, Kolkata, West Bengal 700091

ACKNOWLEDGEMENT

Success of any project depends largely on the encouragement and guidelines of many others. I take this sincere opportunity to express my gratitude to the people who have been instrumental in the successful completion of this project work.

I would like to show our greatest appreciation to ***Mr. Sandipan Gupta***, Project Engineer at Ardent, Kolkata. I always feel motivated and encouraged every time by his valuable advice and constant inspiration; without his encouragement and guidance this project would not have materialized.

Words are inadequate in offering our thanks to the other trainees, project assistants and other members at Ardent Computech Pvt. Ltd. for their encouragement and cooperation in carrying out this project work. The guidance and support received from all the members and who are contributing to this project, was vital for the success of this project.

CONTENTS

OVERVIEW

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and has fewer syntactical constructions than other languages.

Python is interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to Perl and PHP.

Python is Interactive: You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python is Object-Oriented: Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

Python is a Beginner's Language: Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

HISTORY OF PYTHON

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands. Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, Small Talk, UNIX shell, and other scripting languages. Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL). Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

FEATURES OF PYTHON

Easy-to-learn: Python has few Keywords, simple structure and clearly defined syntax. This allows a student to pick up the language quickly.

Easy-to-Read: Python code is more clearly defined and visible to the eyes.

Easy -to-Maintain: Python's source code is fairly easy-to-maintain.

A broad standard library: Python's bulk of the library is very portable and cross platform compatible on UNIX, Windows, and Macintosh.

Interactive Mode: Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.

Portable: Python can run on the wide variety of hardware platforms and has the same interface on all platforms.

Extendable: You can add low level modules to the python interpreter. These modules enables programmers to add to or customize their tools to be more efficient.

Databases: Python provides interfaces to all major commercial databases.

GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries, and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

Scalable: Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below:

- It support functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte code for building large applications.
- It provides very high level dynamic datatypes and supports dynamic type checking.
- It supports automatic garbage collections.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA and JAVA.

BASIC SYNTAX OF PYTHON PROGRAM

Type the following text at the Python prompt and press the Enter –

```
>>> print "Hello, Python!"
```

If you are running new version of Python, then you would need to use print statement with parenthesis as in **print ("Hello, Python!")**
However in Python version 2.4.3, this produces the following result –

```
Hello, Python!
```

Python Identifiers

A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).
Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive programming language.

Python Keywords

The following list shows the Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

And, exec, not
Assert, finally, or
Break, for, pass
Class, from, print
continue, global, raise
def, if, return
del, import, try
elif, in, while
else, is, with
except, lambda, yield

Lines & Indentation

Python provides no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced. The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. For example –

```
if True:
print "True"
else:
print "False"
```

VARIABLE TYPES

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Assigning Values to Variables

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

```
counter=10          # An integer assignment
weight=10.60        # A floating point
name="Ardent"       # A string
```

Multiple Assignment

Python allows you to assign a single value to several variables simultaneously. For example −
a = b = c = 1
a,b,c = 1,2,"hello"

Standard Data Types

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has five standard data types −

- String
- List
- Tuple
- Dictionary
- Number

Data Type Conversion

Sometimes, you may need to perform conversions between the built-in types. To convert between types, you simply use the type name as a function.

There are several built-in functions to perform conversion from one data type to another.

FUNCTIONS

Defining a Function

- def functionname(parameters):
  "function_docstring"
  function_suite
  return [expression]

Pass by reference vs Pass by value

All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function. For example –

```
# Function definition is here

def changeme(mylist):
        "This changes a passed list into this function"
        mylist.append([1,2,3,4]);
        print"Values inside the function: ",mylist
        return

# Now you can call changeme function

mylist=[10,20,30];
changeme(mylist);
print"Values outside the function: ",mylist
```

Here, we are maintaining reference of the passed object and appending values in the same object. So, this would produce the following result −

```
Values inside the function: [10, 20, 30, [1, 2, 3, 4]]
Values outside the function: [10, 20, 30, [1, 2, 3, 4]]
```

Global vs. Local variables

Variables that are defined inside a function body have a local scope, and those defined outside have a global scope . For Example-

```
total=0;                    # This is global variable.

# Function definition is here

def sum( arg1, arg2 ):

# Add both the parameters and return them."

total= arg1 + arg2;       # Here total is local variable.
print"Inside the function local total : ", total
return total;

# Now you can call sum function

sum(10,20);
print"Outside the function global total : ", total
```

When the above code is executed, it produces the following result −

```
Inside the function local total : 30
Outside the function global total : 0
```

MODULES

A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference .

The Python code for a module named *aname* normally resides in a file named *aname.py*. Here's an example of a simple module, support.py

```
def print_func( par ):
        print"Hello : ", par
        return
```

The *import* Statement

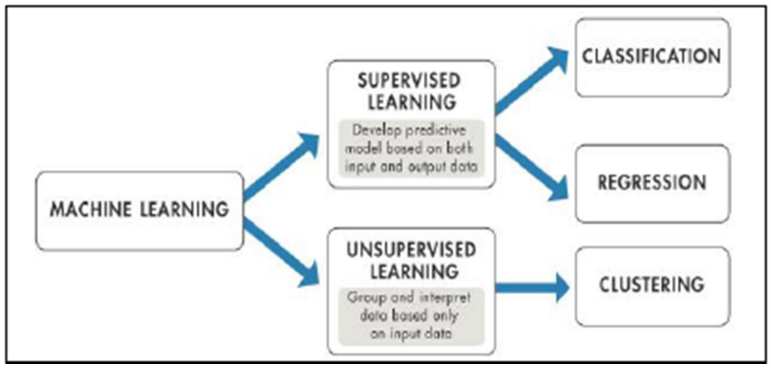
You can use any Python source file as a module by executing an import statement in some other Python source file. The *import* has the following syntax –

```
import module1[, module2[,… moduleN]
```

PACKAGES

A package is a hierarchical file directory structure that defines a single Python application environment that consists of modules and sub packages and sub-subpackages, and so on.

MACHINE LEARNING



Machine learning is a field of computer science that gives computers the ability to learn without being explicitly programmed.
Evolved from the study of pattern recognition and computational learning theory in artificial intelligence, machine learning explores the study and construction of algorithms that can learn from and make predictions on data.

INTRODUCTION TO MACHINE LEARNING

Machine learning is a field of computer science that gives computers the ability to learn without being explicitly programmed.

Arthur Samuel, an American pioneer in the field of computer gaming and artificial intelligence, coined the term "Machine Learning" in 1959 while at IBM. Evolved from the study of pattern recognition and computational learning theory in artificial intelligence, machine learning explores the study and construction of algorithms that can learn from and make predictions on data

Machine learning tasks are typically classified into two broad categories, depending on whether there is a learning "signal" or "feedback" available to a learning system:-

SUPERVISED LEARNING

Supervised learning is the machine learning task of inferring a function from *labeled training data*.[1] The training data consist of a set of *training examples*. In supervised learning, each example is a *pair* consisting of an input object (typically a vector) and a desired output value.

A supervised learning algorithm analyses the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way.

UNSUPERVISED LEARNING

Unsupervised learning is the machine learning task of inferring a function to describe hidden structure from "unlabelled" data (a classification or categorization is not included in the observations). Since the examples given to the learner are unlabelled, there is no evaluation of the accuracy of the structure that is output by the relevant algorithm—which is one way of distinguishing unsupervised learning from supervised learning and reinforcement learning.

A central case of unsupervised learning is the problem of density estimation in statistics, though unsupervised learning encompasses many other problems (and solutions) involving summarizing and explaining key features of the data.

NUMPY

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin.

NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents.

Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted, and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars.

NUMPY ARRAY

NumPy's main object is the homogeneous multidimensional array. It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. In NumPy dimensions are called *axes*. The number of axes is *rank*.

For example, the coordinates of a point in 3D space [1, 2, 1] is an array of rank 1, because it has one axis. That axis has a length of 3. In the example pictured below, the array has rank 2 (it is 2-dimensional). The first dimension (axis) has a length of 2, the second dimension has a length of 3.

[[1., 0., 0.],
 [0., 1., 2.]]

NumPy's array class is called *ndarray*. It is also known by the alias.

SLICING NUMPY ARRAY

import numpy as np

a = np.array([[1,2,3],[3,4,5],[4,5,6]])

```
print 'Our array is:'
print a
print '\n'


print 'The items in the second column are:'
print a[...,1]
print '\n'


print 'The items in the second row are:'
print a[1,...]
print '\n'


print 'The items column 1 onwards are:'
print a[...,1:]
```

OUTPUT

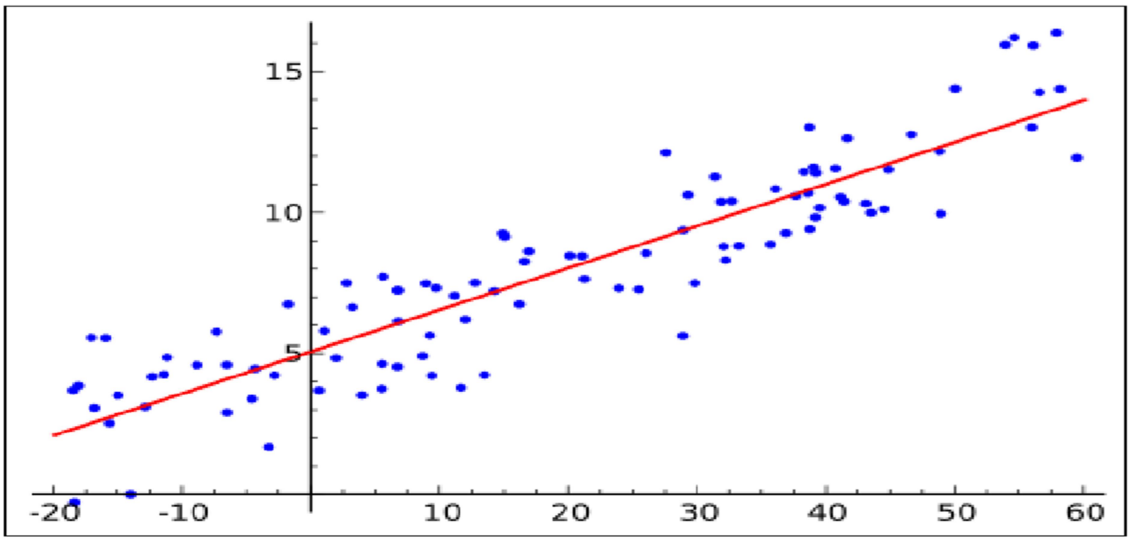
```
Our array is:
[[1 2 3]
[3 4 5]
[4 5 6]]

The items in the second column are:
[2 4 5]

The items in the second row are:
[3 4 5]

The items column 1 onwards are:
[[2 3]
[4 5]
[5 6]]
```

SCIKIT-LEARN

Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, *k*-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

The scikit-learn project started as scikits.learn, a Google Summer of Code project by David Cournapeau. Its name stems from the notion that it is a "SciKit" (SciPy Toolkit), a separately-developed and distributed third-party extension to SciPy.[4] The original codebase was later rewritten by other developers. In 2010 Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort and Vincent Michel, all from INRIA took leadership of the project and made the first public release on February the 1st 2010[5]. Of the various scikits, scikit-learn as well as scikit-image were described as "well-maintained and popular" in November 2012.

REGRESSION ANALYSIS



In statistical modelling, **regression analysis** is a set of statistical processes for estimating the relationships among variables. It includes many techniques for modelling and analysing several variables, when the focus is on the relationship between a dependent variable and one or more independent variables (or 'predictors'). More specifically, regression analysis helps one understand how the typical value of the dependent variable (or 'criterion variable') changes when any one of the independent variables is varied, while the other independent variables are held fixed.

Regression analysis is widely used for prediction and forecasting, where its use has substantial overlap with the field of machine learning. Regression analysis is also used to understand which among the independent variables are related to the dependent variable, and to explore the forms of these relationships. In restricted circumstances, regression analysis can be used to infer casual relationships between the independent and dependent variables. However this can lead to illusions or false relationships, so caution is advisable

LINEAR REGRESSION

Linear regression is a linear approach for modelling the relationship between a scalar dependent variable y and one or more explanatory variables (or independent variables) denoted X. The case of one explanatory variable is called *simple linear regression*. For more than one explanatory variable, the process is called *multiple linear regression*.

In linear regression, the relationships are modelled using linear predictor functions whose unknown model parameters are estimated from the data. Such models are called *linear models*.

LOGISTIC REGRESSION

Logistic regression, or logit regression, or logit model [1] is a regression model where the dependent variable (DV) is categorical. This article covers the case of a binary dependent variable—that is, where the output can take only two values, "0" and "1", which represent outcomes such as pass/fail, win/lose, alive/dead or healthy/sick. Cases where the dependent variable has more than two outcome categories may be analysed in multinomial logistic regression, or, if the multiple categories are

ordered, in ordinal logistic regression. In the terminology of economics, logistic regression is an example of a qualitative response/discrete choice model.

POLYNOMIAL REGRESSION

Polynomial regression is a form of regression analysis in which the relationship between the independent variable x and the dependent variable y is modelled as an n^{th} degree polynomial in x.

Polynomial regression fits a nonlinear relationship between the value of x and the corresponding conditional mean of y , denoted E(y | x), and has been used to describe nonlinear phenomena such as the growth rate of tissues, the distribution of carbon isotopes in lake sediments, and the progression of disease epidemics.

Although *polynomial regression* fits a nonlinear model to the data, as a statistical estimation problem it is linear, in the sense that the regression function E(y | x) is linear in the unknown parameters that are estimated from the data.

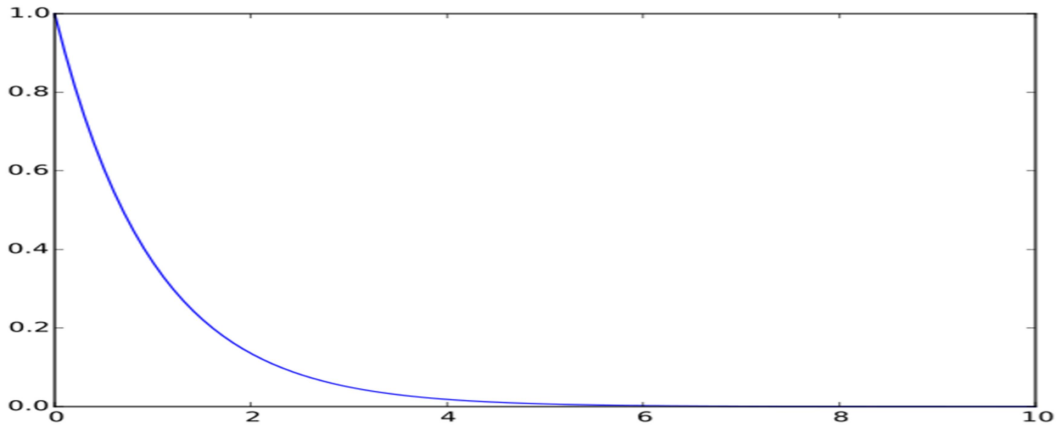
MATPLOTLIB

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter,wxPython, Qt, or GTK+. There is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged .SciPy makes use of matplotlib.

EXAMPLE

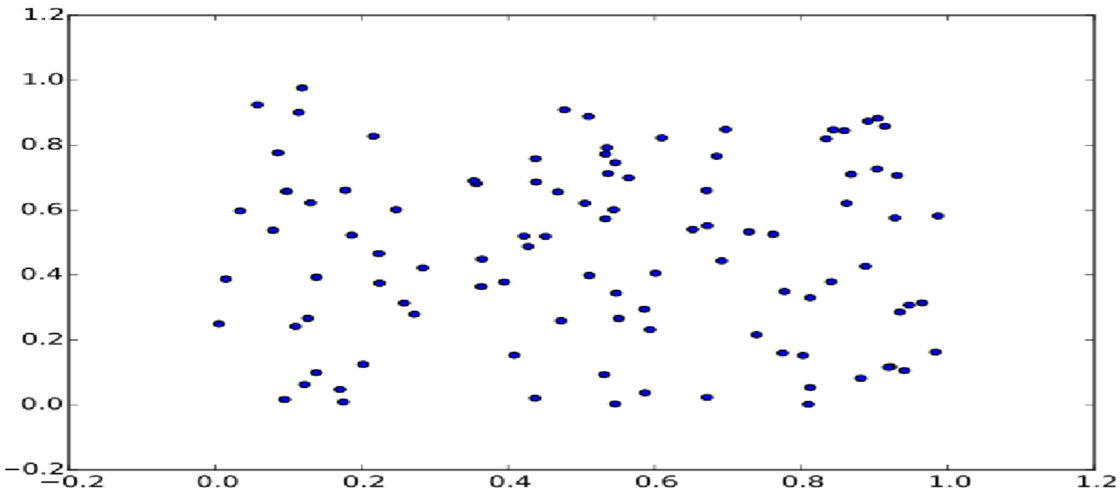
- **LINE PLOT**

```
>>>import matplotlib.pyplot as plt
>>>import numpy as np
>>> a =np.linspace(0,10,100)
>>> b =np.exp(-a)
>>>plt.plot(a,b)
>>>plt.show()
```



- **SCATTER PLOT**

```
>>>import matplotlib.pyplot as plt
>>>from numpy.random import rand
>>> a =rand(100)
>>> b =rand(100)
>>>plt.scatter(a,b)
>>>plt.show()
```



PANDAS

In computer programming, **pandas** is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license. "Panel data", an econometrics term for multidimensional, structured data sets.

LIBRARY FEATURES

- Data Frame object for data manipulation with integrated indexing.
- Tools for reading and writing data between in-memory data structures and different file formats.
- Data alignment and integrated handling of missing data.
- Reshaping and pivoting of data sets.
- Label-based slicing, fancy indexing, and sub setting of large data sets.
- Data structure column insertion and deletion.
- Group by engine allowing split-apply-combine operations on data sets.
- Data set merging and joining.
- Hierarchical axis indexing to work with high-dimensional data in a lower-dimensional data structure.
- Time series-functionality: Date range generation.

DECISION TREE

A **decision tree** is a **decision** support tool that uses a **tree**-like graph or model of **decisions** and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements.

A decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.

In decision analysis, a decision tree and the closely related influence diagram are used as a visual and analytical decision support tool, where the expected values (or expected utility) of competing alternatives are calculated.

A decision tree consists of three types of nodes:[1]

1. Decision nodes – typically represented by squares
2. Chance nodes – typically represented by circles
3. End nodes – typically represented by triangles

Decision trees are commonly used in operations research and operations management. If, in practice, decisions have to be taken online with no recall under incomplete knowledge, a decision tree should be paralleled by a probability model as a best choice model or online selection model algorithm. Another use of decision trees is as a descriptive means for calculating conditional probabilities.

Decision trees, influence diagrams, utility functions, and other decision analysis tools and methods are taught to undergraduate students in schools of business, health economics, and public health, and are examples of operations research or management sciencemethods.

CLUSTERING

Cluster analysis or **clustering** is the task of grouping a set of objects in such a way that objects in the same group (called a **cluster**) are more similar (in some sense or another) to each other than to those in other groups (clusters). It is a main task of exploratory data mining, and a common technique for statistical data analysis, used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, bioinformatics, data compression, and computer graphics.

Cluster analysis itself is not one specific algorithm, but the general task to be solved. It can be achieved by various algorithms that differ significantly in their notion of what constitutes a cluster and how to efficiently find them. Popular notions of clusters include groups with small distances among the cluster members, dense areas of the data space, intervals or particular statistical distributions. Clustering can therefore be formulated as a multi-objective optimization problem.

The appropriate clustering algorithm and parameter settings (including values such as the distance function to use, a density threshold or the number of expected clusters) depend on the individual data set and intended use of the results. Cluster analysis as such is not an automatic task, but an iterative process of knowledge discovery or interactive multi-objective optimization that involves trial and failure. It is often necessary to modify data pre-processing and model parameters until the result achieves the desired properties.

ALGORITHM

- Data Collection
- Data Formatting
- Model Selection
- Training
- Testing

Data Collection: We have prepared data sets of Diabetes & Kidney Disease in .csv format. The datasheet contains the suitable informations and attributes required for the analysis.

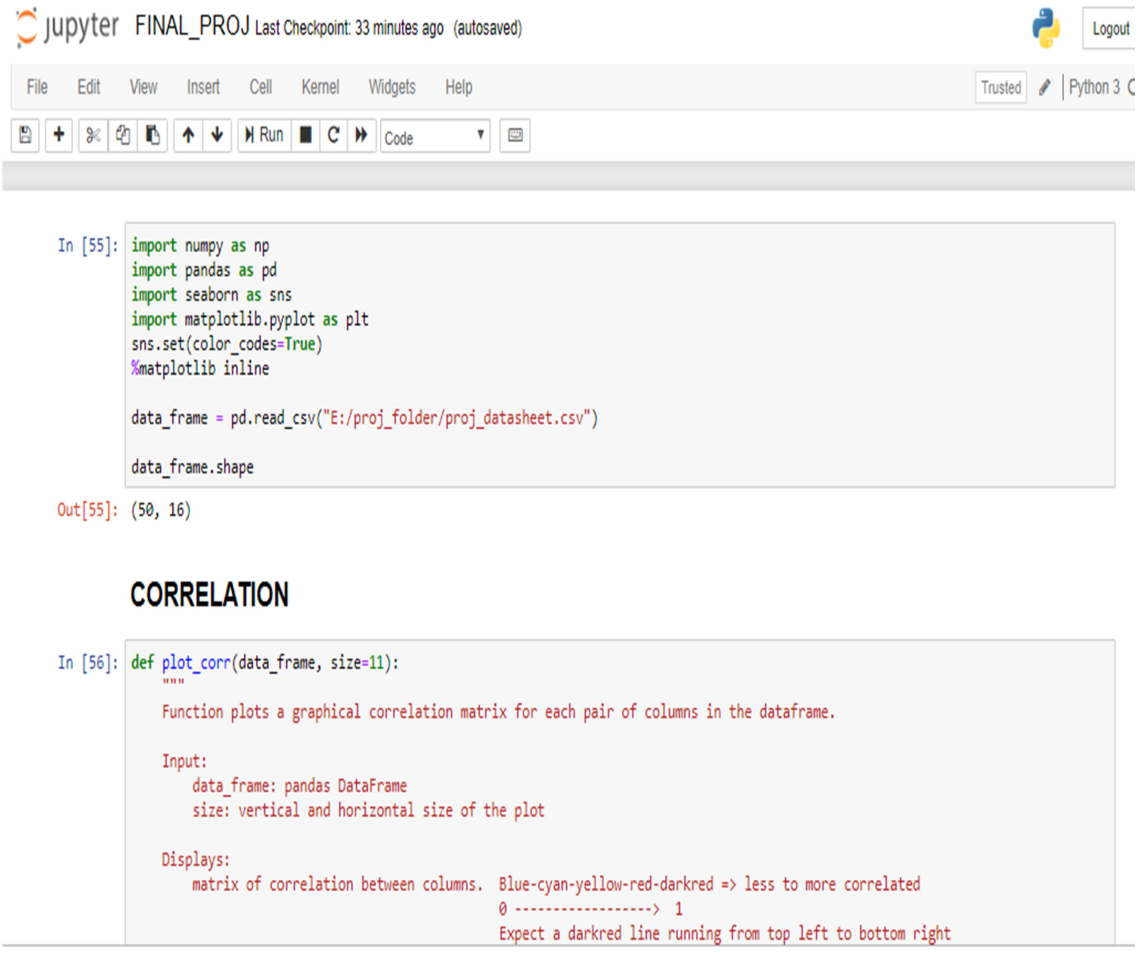
Data Formatting: The data is formatted into suitable data sets. We check the correlation. Algorithms are applied on the specific datasets upon which the disease can be predicted nicely and smoothly.

Model Selection: We have selected different models to minimize the error of the predicted value. The different models used are Linear Regression Model, Logistic Regression Model, Decision Tree Model and K-Means (Clustering) Model.

Training: The data sets were splitted into training and testing sets, such that x_train is used to train the model with corresponding x_test values and some y_train kept reserved for testing.

Testing: The model was tested with y_train and stored in y_pred . Both y_train and y_pred were compared.

ACTUAL CODES FOR DISEASE PREDICTION



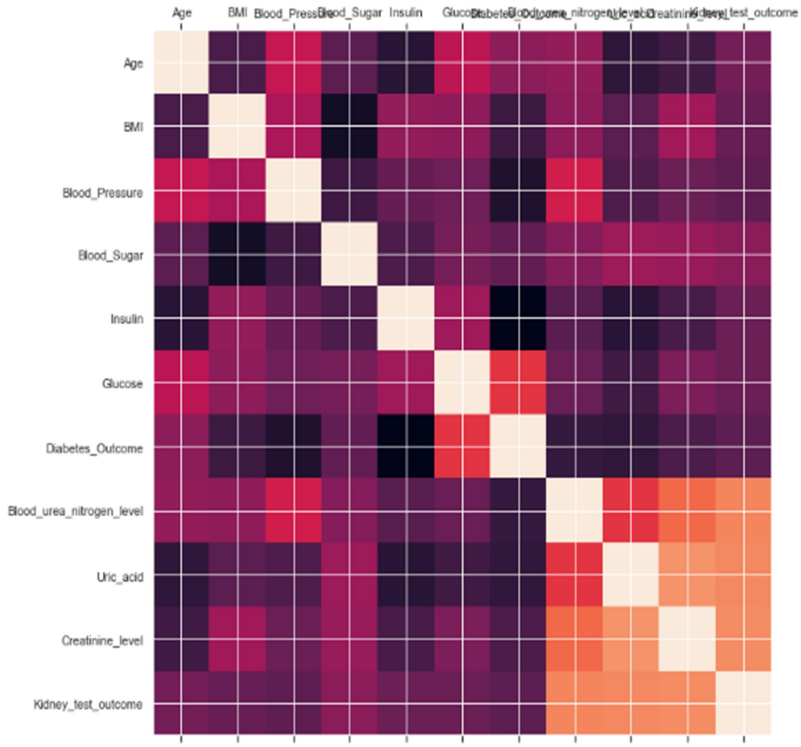
```
                                        Expect a darkred line running from top left to bottom right
    """

    corr = data_frame.corr()    # data frame correlation function
    fig, ax = plt.subplots(figsize=(size, size))
    ax.matshow(corr)   # color code the rectangles by correlation value
    plt.xticks(range(len(corr.columns)), corr.columns)  # draw x tick marks
    plt.yticks(range(len(corr.columns)), corr.columns)  # draw y tick marks
```

In [57]:
```
data_frame.corr()
```

Out[57]:

	Age	BMI	Blood_Pressure	Blood_Sugar	Insulin	Glucose	Diabetes_Outcome	Blood_urea_nitrogen_level	Uric_acid	C
Age	1.000000	0.005466	0.363865	0.063184	-0.098943	0.343225	0.199776	0.222279	-0.075619	
BMI	0.005466	1.000000	0.295682	-0.170338	0.221143	0.206208	-0.033682	0.204468	0.054969	
Blood_Pressure	0.363865	0.295682	1.000000	-0.029250	0.085568	0.121625	-0.127110	0.403617	0.021752	
Blood_Sugar	0.063184	-0.170338	-0.029250	1.000000	0.012649	0.139285	0.080412	0.181586	0.248508	
Insulin	-0.098943	0.221143	0.085568	0.012649	1.000000	0.257081	-0.234672	0.052369	-0.098805	
Glucose	0.343225	0.206208	0.121625	0.139285	0.257081	1.000000	0.477320	0.099939	-0.026066	
Diabetes_Outcome	0.199776	-0.033682	-0.127110	0.080412	-0.234672	0.477320	1.000000	-0.053565	-0.074559	
Blood_urea_nitrogen_level	0.222279	0.204468	0.403617	0.181586	0.052369	0.099939	-0.053565	1.000000	0.480021	
Uric_acid	-0.075619	0.054969	0.021752	0.248508	-0.098805	-0.026066	-0.074559	0.480021	1.000000	
Creatinine_level	-0.023118	0.265964	0.104475	0.237710	0.001285	0.151775	0.015724	0.610877	0.725949	
Kidney_test_outcome	0.132242	0.090342	0.073826	0.190894	0.104956	0.104187	0.065530	0.689735	0.698722	

In [58]:
```
#plotting correlation
plot_corr(data_frame)
```



In [59]:
```
#true cases
num_obs = len(data_frame)
num_true = len(data_frame.loc[data_frame['Diabetes_Outcome'] == 1])
num_false = len(data_frame.loc[data_frame['Diabetes_Outcome'] == 0])
print("Number of True cases:  {0} ({1:2.2f}%)".format(num_true, ((1.00 * num_true)/(1.0 * num_obs)) * 100))
print("Number of False cases: {0} ({1:2.2f}%)".format(num_false, (( 1.0 * num_false)/(1.0 * num_obs)) * 100))
```

```
Number of True cases:  20 (40.00%)
Number of False cases: 30 (60.00%)
```

HISTOGRAMS

In [60]:
```
import pandas as pd
df = pd.read_csv("E:/proj_folder/proj_datasheet.csv")
df.head()
```

Out[60]:

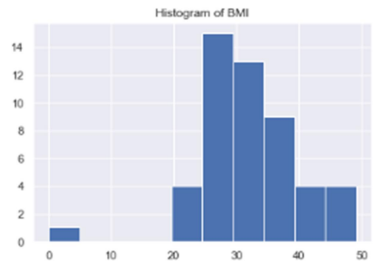
	Serial_ID	Patient_Name	Sex	Blood_group	Age	BMI	Blood_Pressure	Blood_Sugar	Insulin	Glucose	Diabetes_Outcome	Blood_urea_nitrogen_level	U
0	A001	Vidusha Tripathi	Female	AB+	47	33.6	62	80	0	138	1	20	
1	A002	Ram Das	Male	A+	23	38.2	82	52	125	84	0	20	
2	A003	Surya Modak	Male	B+	31	44.2	0	99	0	145	1	7	
3	A004	Jonatham Snow	Male	O+	24	42.3	68	110	250	135	0	15	
4	A005	Motilal Khan	Male	A-	21	40.7	62	155	480	139	0	25	

In [61]:
```
msm= df[['Age','BMI','Blood_Pressure','Insulin','Glucose','Blood_urea_nitrogen_level','Uric_acid','Creatinine_level']]
msm.describe()#it presents the mean,standard deviation,min value,max value of the dataset of patient parameters
```

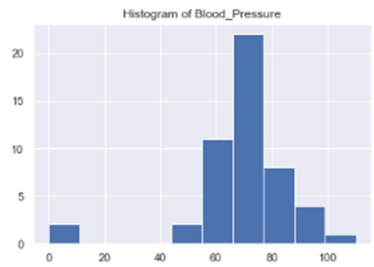
Out[61]:

	Age	BMI	Blood_Pressure	Insulin	Glucose	Blood_urea_nitrogen_level	Uric_acid	Creatinine_level
count	50.00000	50.000000	50.000000	50.000000	50.000000	50.000000	50.000000	50.00000
mean	36.30000	32.064000	68.940000	66.360000	122.800000	23.180000	5.232000	1.34600
std	13.21139	8.178787	17.779155	101.080845	36.027767	8.160857	2.179257	0.73435
min	21.00000	0.000000	0.000000	0.000000	71.000000	5.000000	1.700000	0.00000
25%	25.00000	26.775000	64.000000	0.000000	91.750000	16.500000	3.250000	0.90000
50%	33.00000	32.200000	70.000000	0.000000	114.000000	24.000000	5.650000	1.30000
75%	45.00000	37.275000	77.500000	101.250000	145.750000	29.750000	6.700000	1.57500
max	67.00000	49.100000	110.000000	480.000000	195.000000	38.000000	8.900000	3.60000

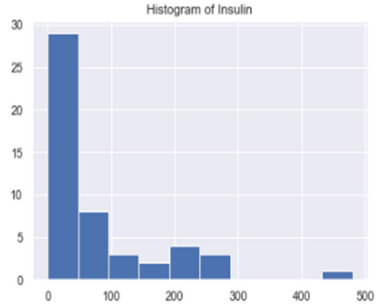
In [62]:
```
#histogram of BMI
import matplotlib.pyplot as plt
import pandas as pd
import pylab as pl
import numpy as np
%matplotlib inline

BMI= df[['BMI',]]
BMI.hist()
plt.title("Histogram of BMI")
plt.show()
```



In [63]:
```
#histogram of blood pressure
bp= df[['Blood_Pressure']]
bp.hist()
plt.title("Histogram of Blood_Pressure")
plt.show()
```

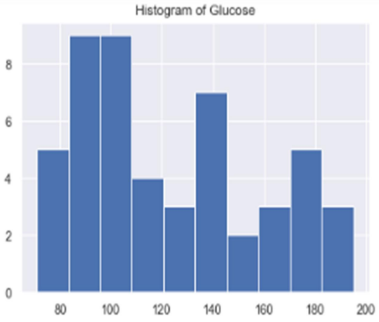


In [64]:
```
#histogram of Insulin
I= df[['Insulin']]
I.hist()
plt.title("Histogram of Insulin")
plt.show()
```

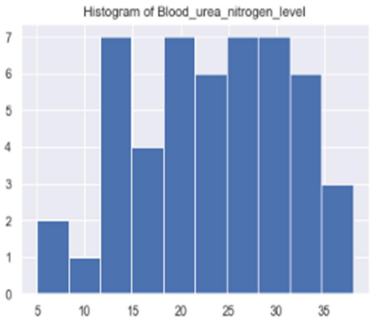


In [65]:
```
#histogram of Glucose
G= df[['Glucose']]
G.hist()
plt.title("Histogram of Glucose")
plt.show()
```

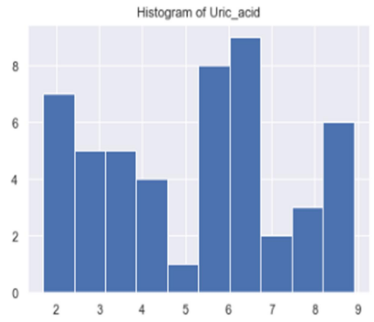
Histogram of Glucose



In [66]:
```
#histogram of Blood Urea Nitrogen Level
bun= df[['Blood_urea_nitrogen_level']]
bun.hist()
plt.title("Histogram of Blood_urea_nitrogen_level")
plt.show()
```

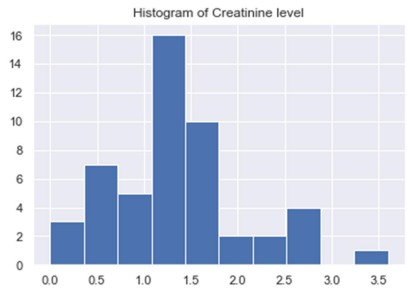


In [67]:
```
#histogram of Uric acid
ua= df[['Uric_acid']]
ua.hist()
plt.title("Histogram of Uric_acid")
plt.show()
```



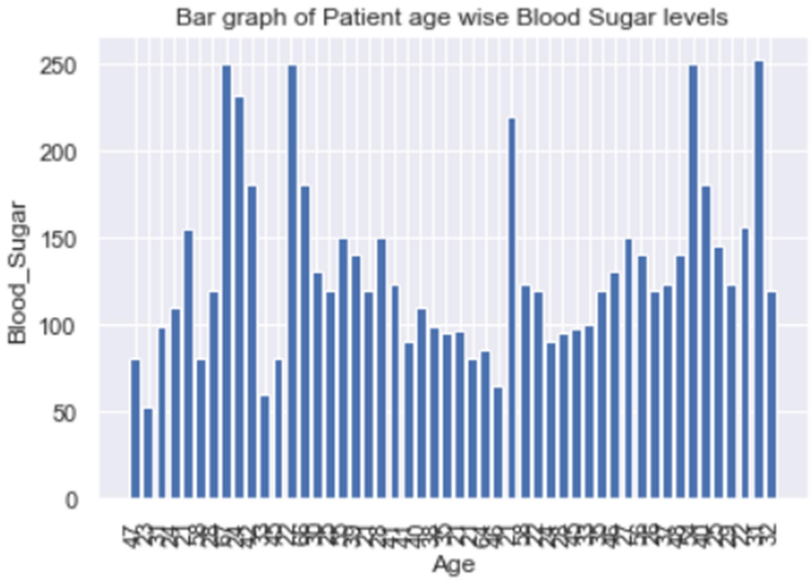
In [68]:
```
#histogram of Creatinine level
c= df[['Creatinine_level']]
c.hist()
plt.title("Histogram of Creatinine level")
plt.show()
```

In [68]:
```
#histogram of Creatinine level
c= df[['Creatinine_level']]
c.hist()
plt.title("Histogram of Creatinine level")
plt.show()
```

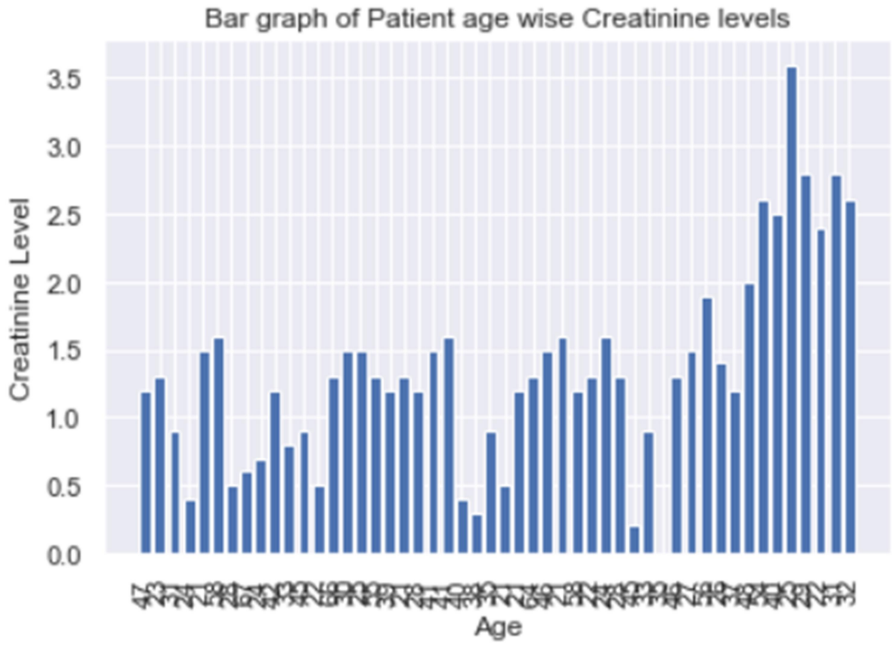


BAR GRAPH

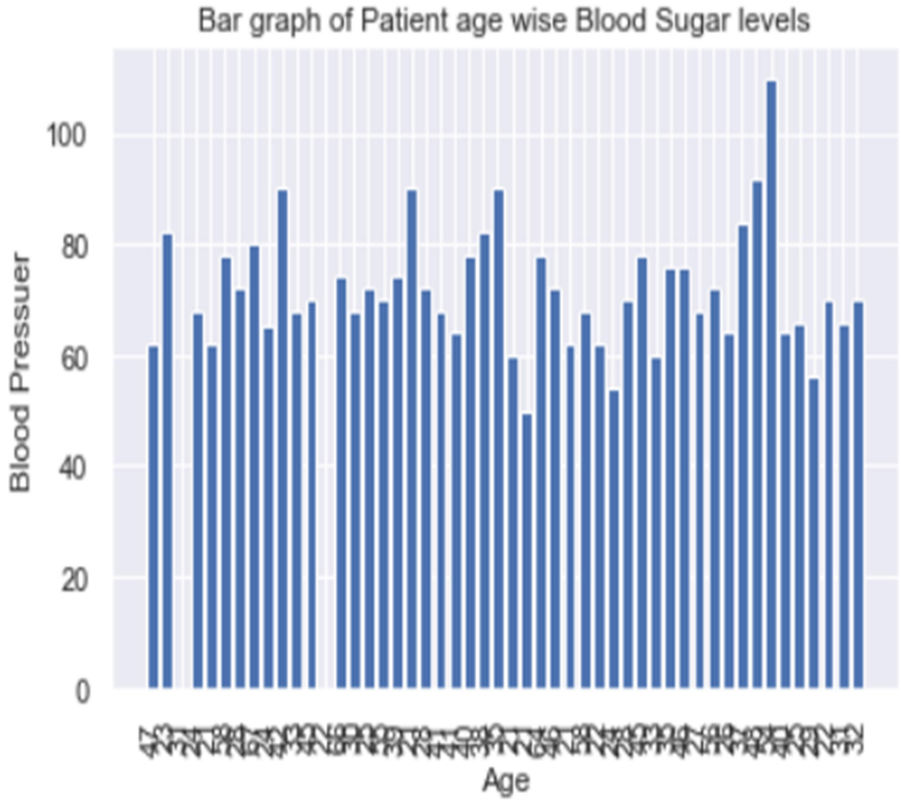
In [69]:
```
#plotting bar graphs

table=pd.read_csv('E:/proj_folder/proj_datasheet.csv')
table.head()

plt.bar(x=np.arange(1,51),height=table['Blood_Sugar'])

plt.title("Bar graph of Patient age wise Blood Sugar levels")

#Give the x axis some labels across the tick marks.
#Argument one is the position for each label
#Argument two is the label values and the final one is to rotate our labels
plt.xticks(np.arange(1,51), table['Age'], rotation=90)

#Give the x and y axes a title
plt.xlabel("Age")
plt.ylabel("Blood_Sugar")

#Finally, show me our new plot
plt.show()
```



In [70]:
```
#plotting bar graphs

table=pd.read_csv('E:/proj_folder/proj_datasheet.csv')
table.head()

plt.bar(x=np.arange(1,51),height=table['Creatinine_level'])

plt.title("Bar graph of Patient age wise Creatinine levels")

#Give the x axis some labels across the tick marks.
#Argument one is the position for each label
#Argument two is the label values and the final one is to rotate our labels
plt.xticks(np.arange(1,51), table['Age'], rotation=90)

#Give the x and y axes a title
plt.xlabel("Age")
plt.ylabel("Creatinine Level")

#Finally, show me our new plot
plt.show()
```



In [71]:
```
#plotting bar graphs

table=pd.read_csv('E:/proj_folder/proj_datasheet.csv')
table.head()

plt.bar(x=np.arange(1,51),height=table['Blood_Pressure'])

plt.title("Bar graph of Patient age wise Blood Sugar levels")

#Give the x axis some labels across the tick marks.
#Argument one is the position for each label
#Argument two is the label values and the final one is to rotate our labels
plt.xticks(np.arange(1,51), table['Age'], rotation=90)

#Give the x and y axes a title
plt.xlabel("Age")
plt.ylabel("Blood Pressuer")

#Finally, show me our new plot
plt.show()
```



LINEAR REGRESSION

FOR DIABETES

In [72]:
```
# Linear Regression

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('E:/proj_folder/proj_datasheet.csv')
X = dataset.iloc[:, [4]].values
y = dataset.iloc[:, 7].values

# Split the DATA into training/testing sets
from sklearn.model_selection import train_test_split
x_train,x_test,y_train, y_test = train_test_split(X,y,test_size=0.2)

# Split the TARGETS into training/testing sets
from sklearn.linear_model import LinearRegression
# Create linear regression object
regr = LinearRegression()

# Train the model using the training sets
regr.fit(x_train, y_train)

# Make predictions using the testing set
y_pred = regr.predict(x_test)

# Plot outputs
plt.scatter(x_train, y_train, color = 'blue')
plt.plot(x_test, y_pred, color = 'black', linewidth = 2)
plt.title('Truth or Bluff (Linear Regression)')
plt.xlabel('Blood Sugar')
plt.ylabel('Diabetes Outcome')
plt.xticks(())
plt.yticks(())
plt.show()
```

In [73]:
```
#accuracy of linear regression
#accuracy of linear regression
import math

from sklearn.metrics import mean_squared_error

rmse = math.sqrt(mean_squared_error(x_test, y_pred))
print(rmse)
#Around 0.001 is great, 1.0 - 2.0 means you should tune your model
#greater than that means if tuning doesn't work, try another model
```

```
94.14013481878024
```

LOGISTIC REGRESSION

FOR DIABETES

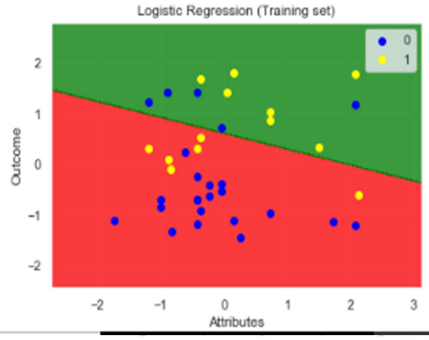
In [74]:
```
from sklearn.model_selection import train_test_split

feature_col_names = ['Blood_Sugar', 'Insulin', 'Glucose']
predicted_class_names = ['Diabetes_Outcome']

X = data_frame[feature_col_names].values     # predictor feature columns (8 X m)
y = data_frame[predicted_class_names].values # predicted class (1=true, 0=false) column (1 X m)
split_test_size = 0.30

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=split_test_size, random_state=42)
                            # test_size = 0.3 is 30%, 42 is the answer to everything
```

In [75]:
```
# Logistic Regression for diabetes

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('E:/proj_folder/proj_datasheet.csv')
X = dataset.iloc[:, [7, 9]].values
y = dataset.iloc[:, 10].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
# Fitting Logistic Regression to the Training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('blue', 'yellow'))(i), label = j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('Attributes')
plt.ylabel('Outcome')
plt.legend()
plt.show()
# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('blue', 'yellow')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],c = ListedColormap(('black', 'white'))(i), label = j)
plt.title('Logistic Regression (Test set)')
plt.xlabel('Attributes')
plt.ylabel('Outcome')
plt.legend()
plt.show()
```

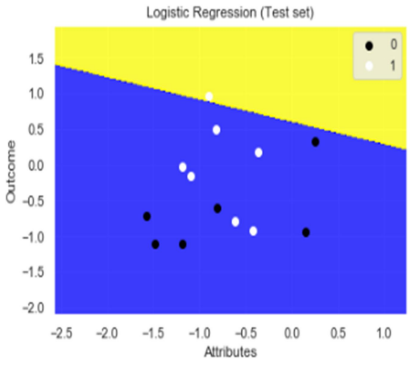
```
# Predicting a new result with Logistic Regression
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

```
C:\Users\TUKAI\Anaconda\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning: Data with input dtype int64 w
as converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\Users\TUKAI\Anaconda\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning: Data with input dtype int64 w
as converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\Users\TUKAI\Anaconda\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning: Data with input dtype int64 w
as converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\Users\TUKAI\Anaconda\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning: Default solver will be changed t
o 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or
RGBA value for all points.
```



Attributes

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or
RGBA value for all points.
```



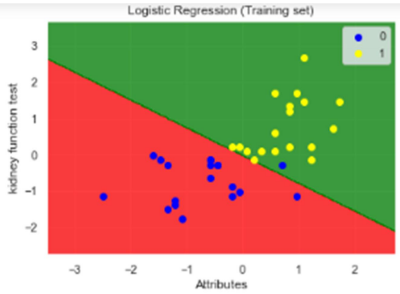
Out[75]: 0.5384615384615384

FOR KIDNEY

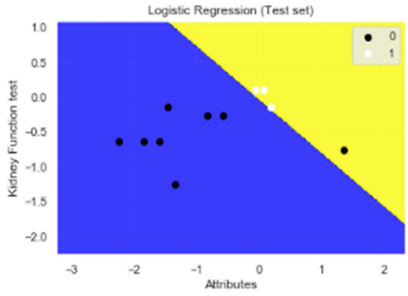
In [76]:
```
# Logistic Regression

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('E:/proj_folder/proj_datasheet.csv')
X = dataset.iloc[:, [11, 13]].values
y = dataset.iloc[:, 14].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Fitting Logistic Regression to the Training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('blue', 'yellow'))(i), label = j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('Attributes')
plt.ylabel('kidney function test')
plt.legend()
plt.show()

# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('blue', 'yellow')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('black', 'white'))(i), label = j)
plt.title('Logistic Regression (Test set)')
plt.xlabel('Attributes')
plt.ylabel('Kidney Function test')
plt.legend()
plt.show()

# Predicting a new result with Logistic Regression
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

```
C:\Users\TUKAI\Anaconda\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning: Default solver will be changed t
o 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or
RGBA value for all points.
```

Logistic Regression (Training set)




```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or
RGBA value for all points.
```



Out[76]: 0.9230769230769231

DECISION TREE

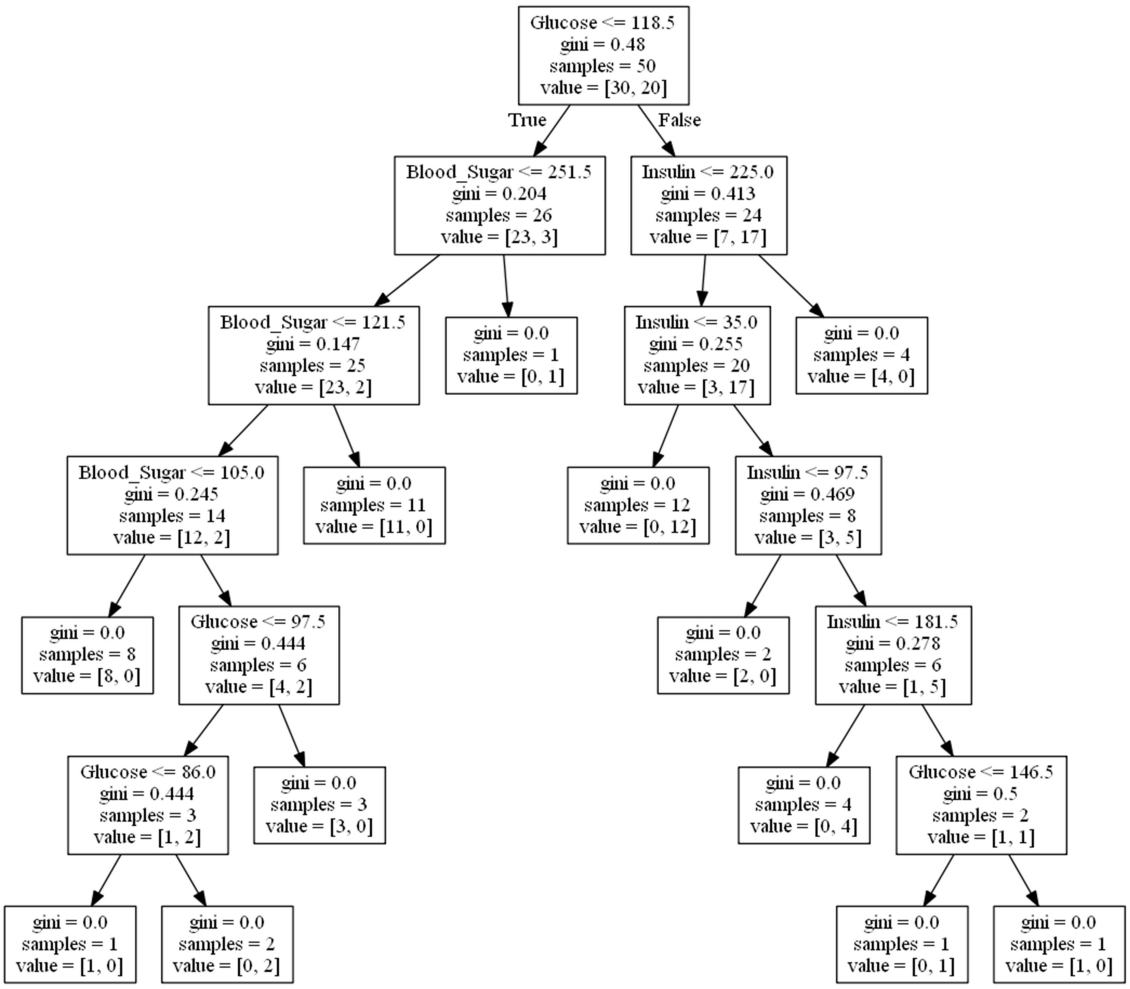
FOR DIABETES

In [89]:
```
feature_col_names = [ 'Blood_Sugar', 'Insulin', 'Glucose']
predicted_class_names = ['Diabetes_Outcome']
X = data_frame[feature_col_names].values     # predictor feature columns (8 X m)
y = data_frame[predicted_class_names].values # predicted class (1=true, 0=false) column (1 X m)
split_test_size = 0.30
from sklearn.model_selection import train_test_split
X_trainset, X_testset, y_trainset, y_testset = train_test_split(X, y, test_size=0.2, random_state=3)
from sklearn.tree import DecisionTreeClassifier
clf=DecisionTreeClassifier(criterion="entropy", max_depth =4)
clf.fit(X,y)
```

Out[89]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=4,
            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')
```

In [90]:
```
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn import tree
import pydotplus
from sklearn.tree import DecisionTreeClassifier
clf=DecisionTreeClassifier()
clf.fit(X,y)
y_pred=clf.predict(X_testset)

from sklearn import metrics
import matplotlib.pyplot as plt
print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_testset, y_pred))
dot_data=StringIO()
tree.export_graphviz(clf,out_file=dot_data,feature_names=feature_col_names)
graph=pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('DIABETES_project.png')
Image=(graph.create_png())
```

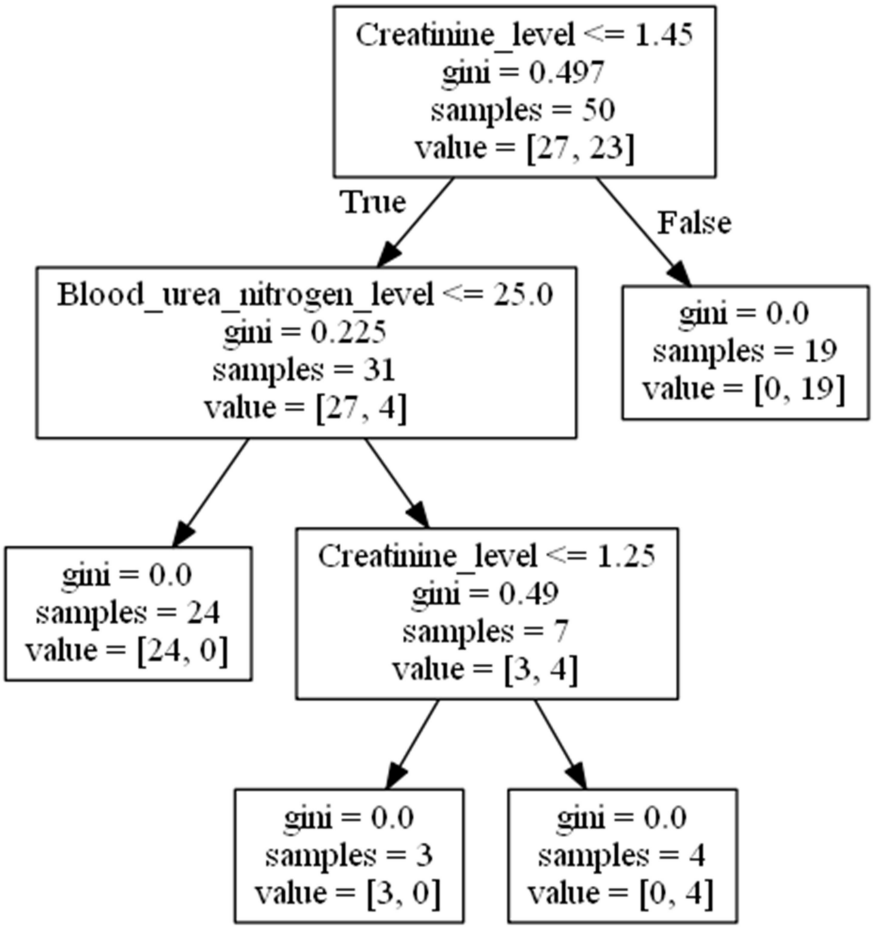
```
DecisionTrees's Accuracy:  1.0
```



FOR KIDNEY

In [91]:
```
from sklearn.model_selection import train_test_split

feature_col_names = [ 'Blood_urea_nitrogen_level', 'Uric_acid', 'Creatinine_level']
predicted_class_names = ['Kidney_test_outcome']

X = data_frame[feature_col_names].values     # predictor feature columns (8 X m)
y = data_frame[predicted_class_names].values # predicted class (1=true, 0=false) column (1 X m)
split_test_size = 0.30
from sklearn.model_selection import train_test_split

X_trainset, X_testset, y_trainset, y_testset = train_test_split(X, y, test_size=0.2, random_state=3)

from sklearn.tree import DecisionTreeClassifier
clf=DecisionTreeClassifier(criterion="entropy", max_depth =4)
clf.fit(X,y)


from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn import tree
import pydotplus

from sklearn.tree import DecisionTreeClassifier
clf=DecisionTreeClassifier()
clf.fit(X,y)
y_pred=clf.predict(X_testset)

from sklearn import metrics
import matplotlib.pyplot as plt
print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_testset, y_pred))


dot_data=StringIO()
tree.export_graphviz(clf,out_file=dot_data,feature_names=feature_col_names)
graph=pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('KIDNEY_project.png')
Image=(graph.create_png())
```

```
DecisionTrees's Accuracy:  1.0
```



In [92]:
```
#decision tree for diabetes medicine
import pandas as pd
import matplotlib.pyplot as plt
import pylab as pl
import numpy as np
%matplotlib inline
df = pd.read_csv("E:/proj_folder/proj_datasheet.csv")
df.head()
```

Out[92]:

	Serial_ID	Patient_Name	Sex	Blood_group	Age	BMI	Blood_Pressure	Blood_Sugar	Insulin	Glucose	Diabetes_Outcome	Blood_urea_nitrogen_level
0	A001	Vidusha Tripathi	Female	AB+	47	33.6	62	80	0	138	1	20
1	A002	Ram Das	Male	A+	23	38.2	82	52	125	84	0	20
2	A003	Surya Modak	Male	B+	31	44.2	0	99	0	145	1	7
3	A004	Jonatham Snow	Male	O+	24	42.3	68	110	250	135	0	15
4	A005	Motilal Khan	Male	A-	21	40.7	62	155	480	139	0	25

In [93]:
```
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier

X =df[['Age', 'Sex','Diabetes_Outcome']].values
d={'Male':1,'Female':0}
df['Sex']=df['Sex'].map(d)
X =df[['Age', 'Sex', 'Glucose','Diabetes_Outcome']].values


y = df["Drug_Diabetes"]

from sklearn.model_selection import train_test_split
X_trainset, X_testset, y_trainset, y_testset = train_test_split(X, y, test_size=0.2, random_state=3)
drugTree = DecisionTreeClassifier(criterion="entropy", max_depth =4)
drugTree # it shows the default parameters
drugTree.fit(X_trainset,y_trainset)
predTree = drugTree.predict(X_testset)
print (predTree [0:50])
print (y_testset [0:50])
```

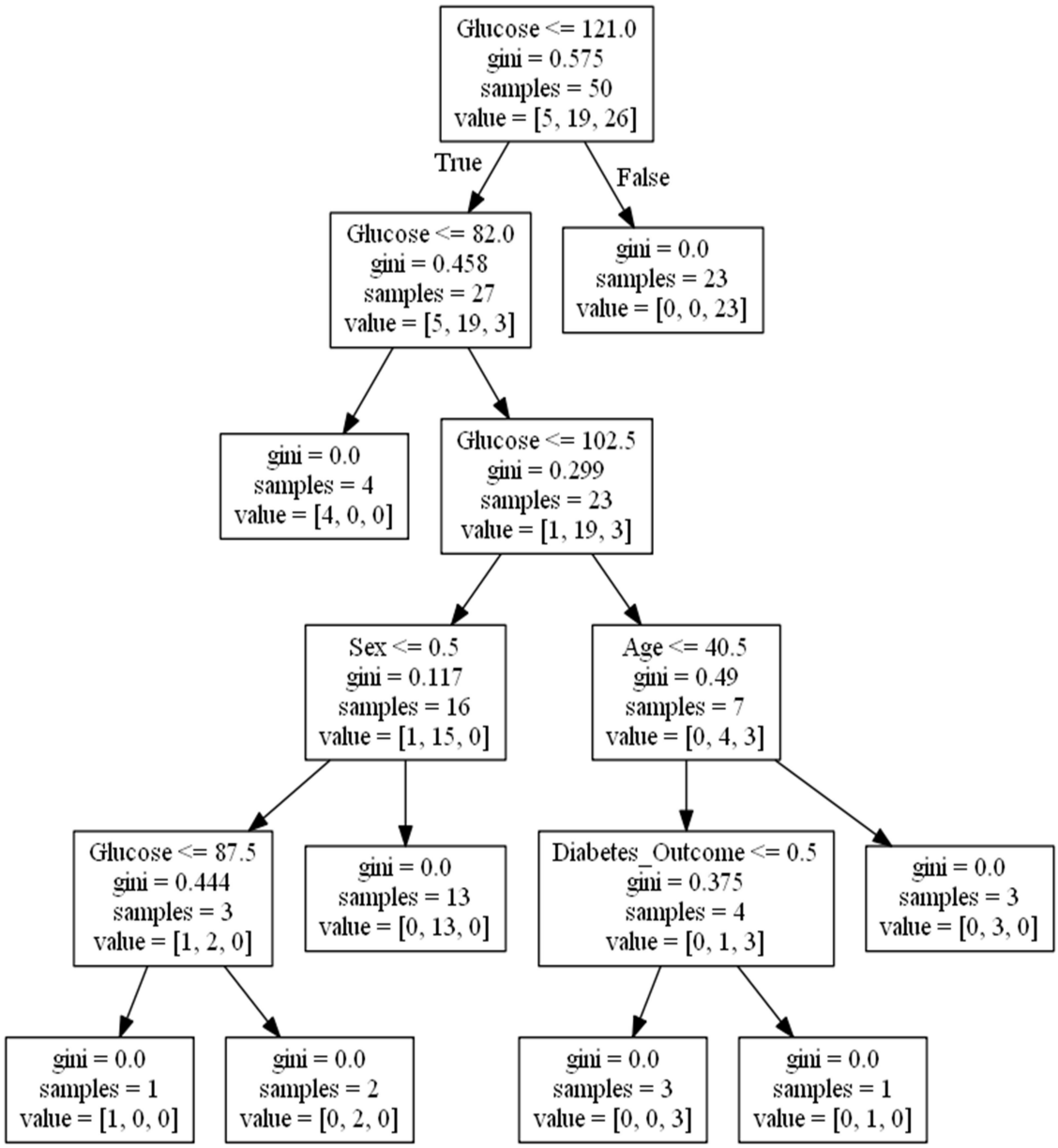
```
from sklearn import metrics
import matplotlib.pyplot as plt
print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_testset, predTree))
```

```
['Actoplus' 'Glyset' 'Glyset' 'Actoplus' 'Glyset' 'Glyset' 'Precose'
 'Precose' 'Precose' 'Glyset']
12    Actoplus
39      Glyset
9       Glyset
47    Actoplus
31      Glyset
28      Glyset
13     Precose
48      Glyset
45     Precose
6       Glyset
Name: Drug_Diabetes, dtype: object
DecisionTrees's Accuracy:  0.9
```

In [95]:
```
from sklearn.externals.six import StringIO
import pydotplus
import matplotlib.image as mpimg
from sklearn import tree
%matplotlib inline

X =df[['Age', 'Glucose','Sex', 'Diabetes_Outcome']]
X[0:50]
feature=list(X)
feature
from sklearn.tree import DecisionTreeClassifier
y=df['Drug_Diabetes']
X=df[feature]
clf=tree.DecisionTreeClassifier()
clf.fit(X,y)
```

Out[95]:
```
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')
```

In [96]:
```
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn import tree
import pydotplus
from sklearn.tree import DecisionTreeClassifier
clf=tree.DecisionTreeClassifier()
clf.fit(X,y)
dot_data=StringIO()
tree.export_graphviz(clf,out_file=dot_data,feature_names=feature)
graph=pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('Drug_Diabetes.png')
Image=(graph.create_png())
```


Glucose <= 121.0
gini = 0.575
samples = 50
value = [5, 19, 26]
True
False
Glucose <= 82.0
gini = 0.458
samples = 27
value = [5, 19, 3]
gini = 0.0
samples = 23
value = [0, 0, 23]
gini = 0.0
samples = 4
value = [4, 0, 0]
Glucose <= 102.5
gini = 0.299
samples = 23
value = [1, 19, 3]
Sex <= 0.5
gini = 0.117
samples = 16
value = [1, 15, 0]
Age <= 40.5
gini = 0.49
samples = 7
value = [0, 4, 3]
Glucose <= 87.5
gini = 0.444
samples = 3
value = [1, 2, 0]
gini = 0.0
samples = 13
value = [0, 13, 0]
Diabetes_Outcome <= 0.5
gini = 0.375
samples = 4
value = [0, 1, 3]
gini = 0.0
samples = 3
value = [0, 3, 0]
gini = 0.0
samples = 1
value = [1, 0, 0]
gini = 0.0
samples = 2
value = [0, 2, 0]
gini = 0.0
samples = 3
value = [0, 0, 3]
gini = 0.0
samples = 1
value = [0, 1, 0]

In [106]:
```
#decision tree for kidney medicine

import pandas as pd
import matplotlib.pyplot as plt
import pylab as pl
import numpy as np
%matplotlib inline
df = pd.read_csv("E:/proj_folder/proj_datasheet.csv")
df.head()
```

Out[106]:

	Serial_ID	Patient_Name	Sex	Blood_group	Age	BMI	Blood_Pressure	Blood_Sugar	Insulin	Glucose	Diabetes_Outcome	Blood_urea_nitrogen_level
0	A001	Vidusha Tripathi	Female	AB+	47	33.6	62	80	0	138	1	20
1	A002	Ram Das	Male	A+	23	38.2	82	52	125	84	0	20
2	A003	Surya Modak	Male	B+	31	44.2	0	99	0	145	1	7
3	A004	Jonatham Snow	Male	O+	24	42.3	68	110	250	135	0	15
4	A005	Motilal Khan	Male	A-	21	40.7	62	155	480	139	0	25

In [107]:
```
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier

X =df[['Age', 'Sex','Diabetes_Outcome']].values
d={'Male':1,'Female':0}
df['Sex']=df['Sex'].map(d)

X =df[['Age', 'Sex', 'Glucose','Diabetes_Outcome']].values


y = df["Drug_kidney"]

from sklearn.model_selection import train_test_split

X_trainset, X_testset, y_trainset, y_testset = train_test_split(X, y, test_size=0.2, random_state=3)

drugTree = DecisionTreeClassifier(criterion="entropy", max_depth =4)
drugTree # it shows the default parameters
```

Out[107]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=4,
            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')
```

In [108]:
```
drugTree.fit(X_trainset,y_trainset)
```

Out[108]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=4,
            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')
```

In [109]:
```
predTree = drugTree.predict(X_testset)
print (predTree [0:50])
print (y_testset [0:50])
```

```
['Enalapril' 'Enalapril' 'Enalapril' 'Enalapril' 'Enalapril' 'Enalapril'
 'Captopril' 'Enalapril' 'Enalapril' 'Enalapril']
12     Ramipril
39    Captopril
9     Enalapril
47    Captopril
31    Enalapril
28    Enalapril
13    Enalapril
48    Captopril
45    Captopril
6     Captopril
Name: Drug_kidney, dtype: object
```

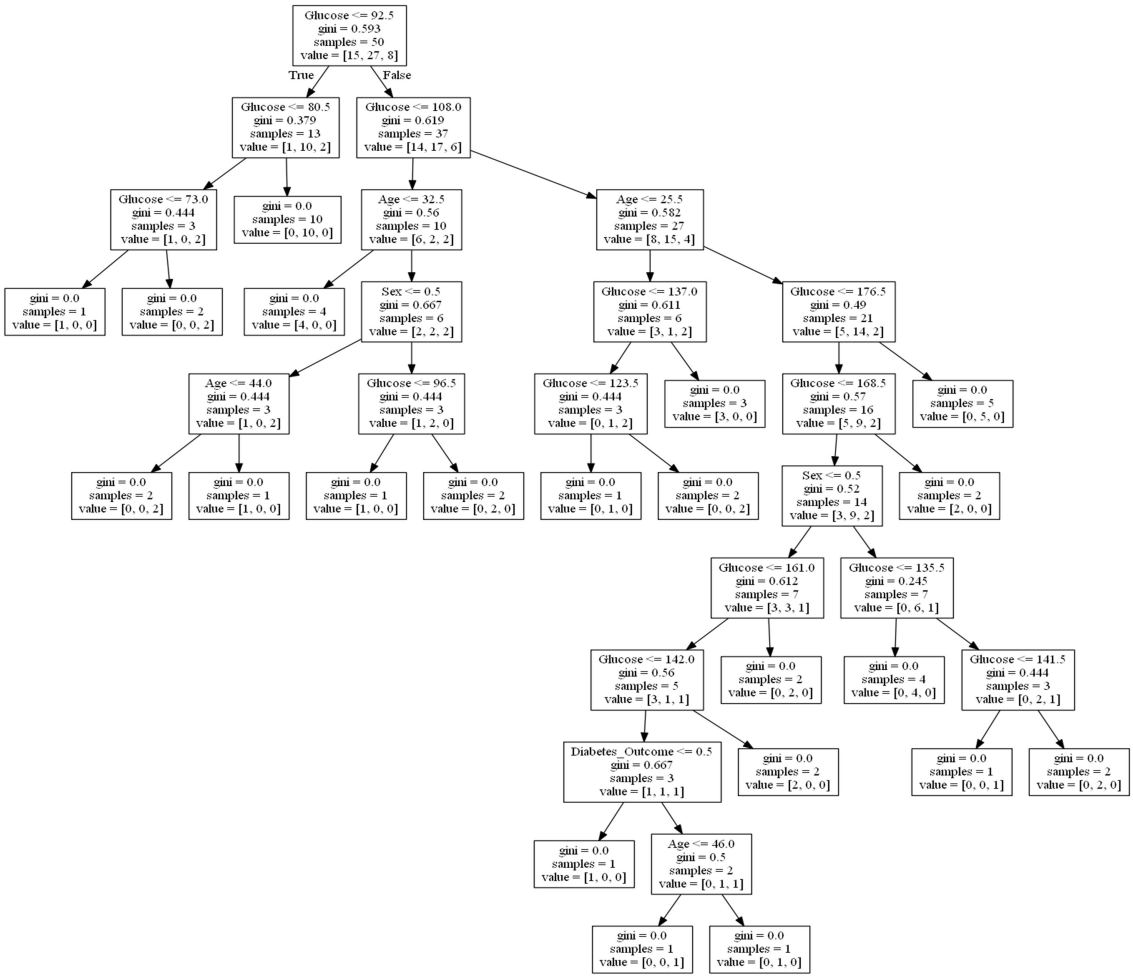
In [110]:
```
from sklearn import metrics
import matplotlib.pyplot as plt
print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_testset, predTree))
```

```
DecisionTrees's Accuracy:  0.3
```

In [111]:
```
from sklearn.externals.six import StringIO
import pydotplus
import matplotlib.image as mpimg
from sklearn import tree
%matplotlib inline

X =df[['Age', 'Glucose','Sex', 'Diabetes_Outcome']]
X[0:50]
feature=list(X)
feature
from sklearn.tree import DecisionTreeClassifier
y=df['Drug_kidney']
X=df[feature]
clf=tree.DecisionTreeClassifier()
clf.fit(X,y)
```

Out[111]:
```
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')
```

In [112]:
```
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn import tree
import pydotplus
from sklearn.tree import DecisionTreeClassifier
clf=tree.DecisionTreeClassifier()
clf.fit(X,y)
dot_data=StringIO()
tree.export_graphviz(clf,out_file=dot_data,feature_names=feature)
graph=pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('Drug_Kidney.png')
Image=(graph.create_png())
```

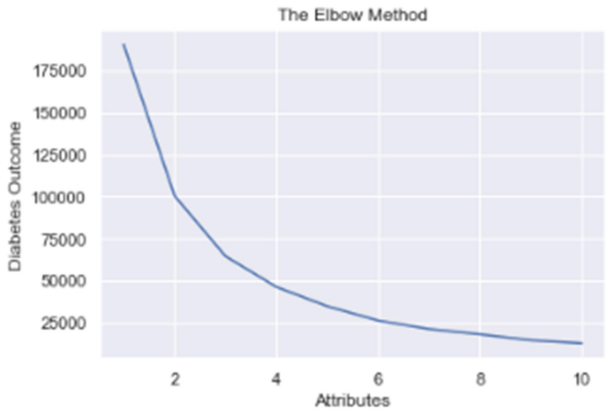


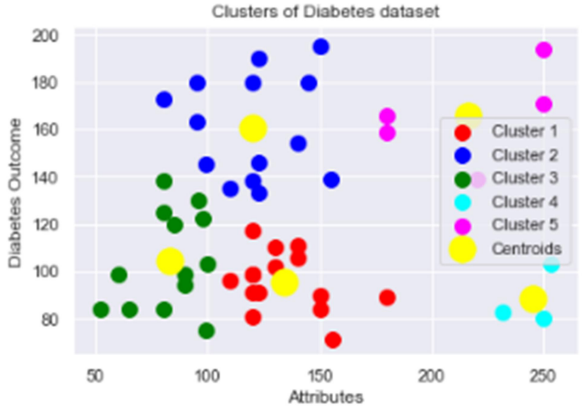
K-Means

FOR DIABETES

In [80]:
```
# K-Means Clustering for diabetes

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

%matplotlib inline

# Importing the dataset
dataset = pd.read_csv('E:/proj_folder/proj_datasheet.csv')
X = dataset.iloc[:, [7, 9]].values
y = dataset.iloc[:, 10].values

# Splitting the dataset into the Training set and Test set
"""from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)"""

# Feature Scaling
"""from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
sc_y = StandardScaler()
y_train = sc_y.fit_transform(y_train)"""

# Using the elbow method to find the optimal number of clusters
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Attributes')
plt.ylabel('Diabetes Outcome')
plt.show()
```

```
# Fitting K-Means to the dataset
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)

# Visualising the clusters
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label = 'Centroids')
plt.title('Clusters of Diabetes dataset')
plt.xlabel('Attributes')
plt.ylabel('Diabetes Outcome')
plt.legend()
plt.show()
```

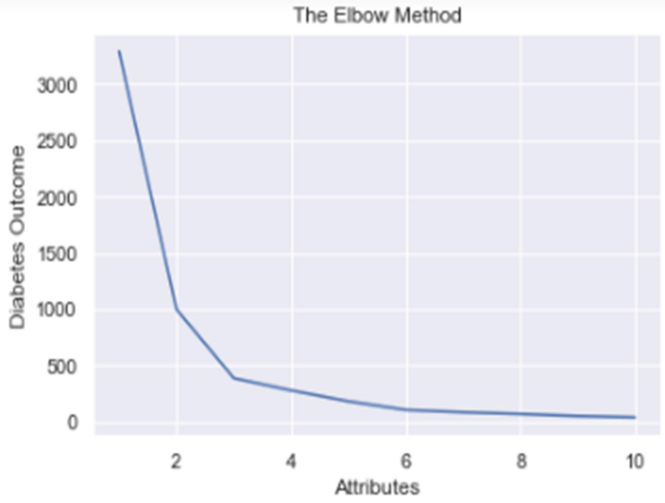


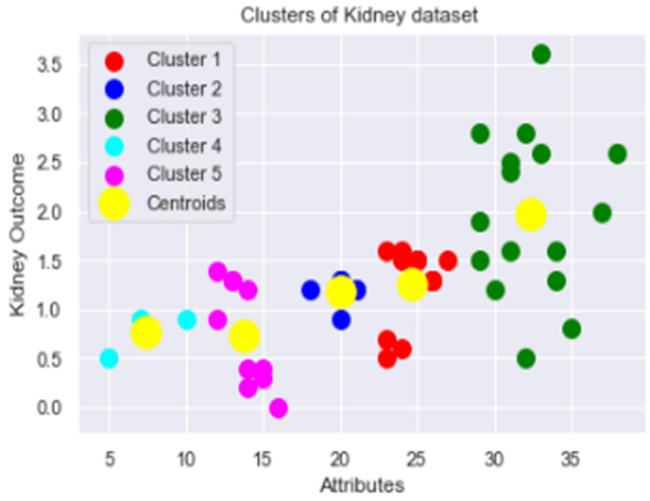


FOR KIDNEY

In [81]:
```
# K-Means Clustering for Kidney disease

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

%matplotlib inline

# Importing the dataset
dataset = pd.read_csv('E:/proj_folder/proj_datasheet.csv')
X = dataset.iloc[:, [11, 13]].values
y = dataset.iloc[:, 14].values

# Splitting the dataset into the Training set and Test set
"""from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)"""

# Feature Scaling
"""from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
sc_y = StandardScaler()
y_train = sc_y.fit_transform(y_train)"""

# Using the elbow method to find the optimal number of clusters
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Attributes')
plt.ylabel('Diabetes Outcome')
plt.show()
```

```
# Fitting K-Means to the dataset
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)

# Visualising the clusters
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label = 'Centroids')
plt.title('Clusters of Kidney dataset ')
plt.xlabel('Attributes')
plt.ylabel('Kidney Outcome')
plt.legend()
plt.show()
```





CONCLUSION

We have collected some of the raw data from healthcare related online sources.

After that we have selected few models for checking the accuracy . We have used four models namely **Linear regression model, Logistic Regression Model, Decision Tree Model and K-Means (Clustering) Model.**

Logistic Regression Model is a good one as accuracy calculated was near about **0.5384 and 0.9230** which is higher than the other models.

FUTURE SCOPE

The data taken was limited. The project could have been extended to more data attributes. The disease was predicted only by taking some of the causes into account, it can also be predicted by looking at the rest of the causes as well.

The error can be minimized as well by using other algorithms.

THANK YOU