Predicting Early Stage Diabetes Risk In Individuals using Machine Learning

Datasource

1. https://archive.ics.uci.edu/ml/datasets/Early+stage+diabetes+risk+prediction+dataset.# (https://archive.ics.uci.edu/ml/datasets/Early+stage+diabetes+risk+prediction+dataset.#) (https://archive.ics.uci.edu/ml/datasets/Early+stage+diabetes+risk+prediction+dataset.# (https://archive.ics.uci.edu/ml/datasets/Early+stage+diabetes+risk+prediction+dataset.#)).
2. https://archive.ics.uci.edu/ml/machine-learning-databases/00529/ (https://archive.ics.uci.edu/ml/machine-learning-databases/00529/) (https://archive.ics.uci.edu/ml/machine-learning-databases/00529/ (https://archive.ics.uci.edu/ml/machine-learning-databases/00529/)).

Project Outline

```
1. Problem
2. Motivation
3. Dataset Information
4. Feature Processing and Feature Engineering
5. Machiine Learning Model Development
6. Prediction/Result
7. Evaluating the result/metrics
8. Conclusion
9. References
```

Problem Statement

```
1 Diabetes is a very common disease with many risk factors that can lead to getting diabetes.
2 Is it possible to predict whether a patient/individual is at a risk ofearly stage diabetes given the signs
asymptoms.
3 Since we are using an already labelled dataset to build a predictive
model our task will be a supervised machine learning problem
4 Therefore we will be using a supervised machine learning classification
approach to solve our problem.
5 Based on the number of target class we have will will need to build a
binary classifier type of ML model.
```

About Dataset

1 Datasource: 2 3 https://archive.ics.uci.edu/ml/datasets/Early+stage+diabetes+risk+predict4456ion+dataset (https://archive.ics.uci.edu/ml/datasets/Early+stage+diabetes+risk+predict4456ion+dataset).

```
5 Description:
The dataset was collected using direct questionnaires from the patients
8
of Sylhet Diabetes Hospital in Sylhet, Bangladesh and approved by a doctor.
9 Metadata:
10
11 The dataset is a multivariate dataset in a CSV format.
12 It has 520 datapoints and 17 fields or attributes.
13 Attribute Information:
14
15 Age 1.20-65
16 Sex 1. Male, 2. Female
17 Polyuria 1. Yes, 2.No.
18 Polydipsia 1.Yes, 2.No.
19 sudden weight loss 1. Yes, 2.No.
20 weakness 1.Yes, 2.No.
21 Polyphagia 1.Yes, 2.No.
22 Genital thrush 1.Yes, 2.No.
23 visual blurring 1.Yes, 2.No.
24 Itching 1. Yes, 2.No.
25 Irritability 1.Yes, 2.No. 26 delayed healing 1. Yes, 2.No. 27 partial paresis 1. Yes, 2.No. 28 muscle
stiness 1.Yes, 2.No. 29 Alopecia 1.Yes, 2.No.
30 Obesity 1.Yes, 2.No.
31 Class 1. Positive, 2. Negative.
```

In [1]:

```
1 # Load EDA Packages
import pandas as pd
import numpy as np
```

In [2]:

```
 # Load Data Viz Packages
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:

```
import warnings
warnings.filterwarnings("ignore")
```

In [4]:

```
 # Load Machine Learning Packages
import sklearn
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

# For Metrics
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
```

In [5]:

```
print("Pandas", pd.__version__)
print("Numpy", np.__version__)
print("Seaborn", sns.__version__)
print("Sklearn", sklearn.__version__)
```

```
Pandas 1.4.4
Numpy 1.21.5
Seaborn 0.11.2
Sklearn 1.0.2
```

Descriptive Analysis of Dataset

In [6]:

```
df=pd.read_csv(r"C:\Users\KALPANA\Downloads\diabetes_data_upload.csv")
```

In [7]:

```
df.head()
```

Out[7]:

	Age	Gender	Polyuria	Polydipsia	sudden weight loss	weakness	Polyphagia	Genital thrush	visual blurring	Itching	Irritability	delayed healing	partial paresis	n sti
0	40	Male	No	Yes	No	Yes	No	No	No	Yes	No	Yes	No	
1	58	Male	No	No	No	Yes	No	No	Yes	No	No	No	Yes	
2	41	Male	Yes	No	No	Yes	Yes	No	No	Yes	No	Yes	No	
3	45	Male	No	No	Yes	Yes	Yes	Yes	No	Yes	No	Yes	No	
4	60	Male	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	

In [8]:

```
df.shape
```

Out[8]:

```
(520, 17)
```

In [9]:

```
df.describe()
```

Out[9]:

	Age
count	520.000000
mean	48.028846
std	12.151466
min	16.000000
25%	39.000000
50%	47.500000
75%	57.000000
max	90.000000

In [10]:

```
df.isnull().sum()
```

Out[10]:

```
Age                   0
Gender                0
Polyuria              0
Polydipsia            0
sudden weight loss    0
weakness              0
Polyphagia            0
Genital thrush        0
visual blurring       0
Itching               0
Irritability          0
delayed healing       0
partial paresis       0
muscle stiffness      0
Alopecia              0
Obesity               0
class                 0
dtype: int64
```

In [11]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 520 entries, 0 to 519
Data columns (total 17 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   Age                 520 non-null    int64 
 1   Gender              520 non-null    object
 2   Polyuria            520 non-null    object
 3   Polydipsia          520 non-null    object
 4   sudden weight loss  520 non-null    object
 5   weakness            520 non-null    object
 6   Polyphagia          520 non-null    object
 7   Genital thrush      520 non-null    object
 8   visual blurring     520 non-null    object
 9   Itching             520 non-null    object
 10  Irritability        520 non-null    object
 11  delayed healing     520 non-null    object
 12  partial paresis     520 non-null    object
 13  muscle stiffness    520 non-null    object
 14  Alopecia            520 non-null    object
 15  Obesity             520 non-null    object
 16  class               520 non-null    object
dtypes: int64(1), object(16)
memory usage: 69.2+ KB
```

```
Observation
2 There are no missing values and we have 520 datapoints and 17 Columns
3 Most of the columns/fields are of the Object type we will need to convert
them to a proper format

```

```
1 Data Cleaning
2
3 Convert the column names to a better case and format
4 Encode the dataset into numeric format using either LabelEncoder or
Custom Function
5 Gender: Female(0), Male(1)
6 all
No (0), Yes (1)

```

In [12]:

```
1 # Converting the columns
df.columns.str.lower().str.replace(" ","_")
```

Out[12]:

```
Index(['age', 'gender', 'polyuria', 'polydipsia', 'sudden_weight_loss',
       'weakness', 'polyphagia', 'genital_thrush', 'visual_blurring',
       'itching', 'irritability', 'delayed_healing', 'partial_paresis',
       'muscle_stiffness', 'alopecia', 'obesity', 'class'],
      dtype='object')
```

In [13]:

```
 # Converting the columns
df.columns=df.columns.str.lower().str.replace(" ","_")
```

In [14]:

```
df.columns
```

Out[14]:

```
Index(['age', 'gender', 'polyuria', 'polydipsia', 'sudden_weight_loss',
       'weakness', 'polyphagia', 'genital_thrush', 'visual_blurring',
       'itching', 'irritability', 'delayed_healing', 'partial_paresis',
       'muscle_stiffness', 'alopecia', 'obesity', 'class'],
      dtype='object')
```

In [15]:

```
# Encode the dataset
from sklearn.preprocessing import LabelEncoder
```

In [16]:

```
df.select_dtypes (include='object').columns
```

Out[16]:

```
Index(['gender', 'polyuria', 'polydipsia', 'sudden_weight_loss', 'weakness',
       'polyphagia', 'genital_thrush', 'visual_blurring', 'itching',
       'irritability', 'delayed_healing', 'partial_paresis',
       'muscle_stiffness', 'alopecia', 'obesity', 'class'],
      dtype='object')
```

In [17]:

```
objList = df.select_dtypes (include='object').columns
```

In [18]:

```
objList
```

Out[18]:

```
Index(['gender', 'polyuria', 'polydipsia', 'sudden_weight_loss', 'weakness',
       'polyphagia', 'genital_thrush', 'visual_blurring', 'itching',
       'irritability', 'delayed_healing', 'partial_paresis',
       'muscle_stiffness', 'alopecia', 'obesity', 'class'],
      dtype='object')
```

In [19]:

```
columns_to_label_encode=[ 'polyuria', 'polydipsia', 'sudden_weight_loss', 'weakness',
       'polyphagia', 'genital_thrush', 'visual_blurring', 'itching',
       'irritability', 'delayed_healing', 'partial_paresis',
       'muscle_stiffness', 'alopecia', 'obesity']
```

In [20]:

```
columns_to_label_encode
```

Out[20]:

```
['polyuria',
 'polydipsia',
 'sudden_weight_loss',
 'weakness',
 'polyphagia',
 'genital_thrush',
 'visual_blurring',
 'itching',
 'irritability',
 'delayed_healing',
 'partial_paresis',
 'muscle_stiffness',
 'alopecia',
 'obesity']
```

In [21]:

```
LE=LabelEncoder()
```

In [22]:

```
# df["Polyuria"] = LE.fit_transform(df["Polyuria"].astype(str))
```

In [23]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 520 entries, 0 to 519
Data columns (total 17 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   age                 520 non-null    int64 
 1   gender              520 non-null    object
 2   polyuria            520 non-null    object
 3   polydipsia          520 non-null    object
 4   sudden_weight_loss  520 non-null    object
 5   weakness            520 non-null    object
 6   polyphagia          520 non-null    object
 7   genital_thrush      520 non-null    object
 8   visual_blurring     520 non-null    object
 9   itching             520 non-null    object
 10  irritability        520 non-null    object
 11  delayed_healing     520 non-null    object
 12  partial_paresis     520 non-null    object
 13  muscle_stiffness    520 non-null    object
 14  alopecia            520 non-null    object
 15  obesity             520 non-null    object
 16  class               520 non-null    object
dtypes: int64(1), object(16)
memory usage: 69.2+ KB
```

In [24]:

```
# Encode Every column except age, gender and class
for col in columns_to_label_encode:
    print(col)
```

```
polyuria
polydipsia
sudden_weight_loss
weakness
polyphagia
genital_thrush
visual_blurring
itching
irritability
delayed_healing
partial_paresis
muscle_stiffness
alopecia
obesity
```

In [25]:

```
# Encode Every column except age, gender and class
for col in columns_to_label_encode:
    df[col] = LE.fit_transform(df[col].astype(str))

```

In [26]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 520 entries, 0 to 519
Data columns (total 17 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   age                 520 non-null    int64 
 1   gender              520 non-null    object
 2   polyuria            520 non-null    int32 
 3   polydipsia          520 non-null    int32 
 4   sudden_weight_loss  520 non-null    int32 
 5   weakness            520 non-null    int32 
 6   polyphagia          520 non-null    int32 
 7   genital_thrush      520 non-null    int32 
 8   visual_blurring     520 non-null    int32 
 9   itching             520 non-null    int32 
 10  irritability        520 non-null    int32 
 11  delayed_healing     520 non-null    int32 
 12  partial_paresis     520 non-null    int32 
 13  muscle_stiffness    520 non-null    int32 
 14  alopecia            520 non-null    int32 
 15  obesity             520 non-null    int32 
 16  class               520 non-null    object
dtypes: int32(14), int64(1), object(2)
memory usage: 40.8+ KB
```

In [27]:

```
df.head()
```

Out[27]:

	age	gender	polyuria	polydipsia	sudden_weight_loss	weakness	polyphagia	genital_thrush	visual_blurring	itching	irritabil
0	40	Male	0	1	0	1	0	0	0	1	
1	58	Male	0	0	0	1	0	0	1	0	
2	41	Male	1	0	0	1	1	0	0	1	
3	45	Male	0	0	1	1	1	1	0	1	
4	60	Male	1	1	1	1	1	0	1	1	

In [28]:

```
# List Initial Classes
print (LE.classes_)
```

```
['No' 'Yes']
```

In [29]:

```
# Method 2 Using Custom Function for encoding gender and class columns
df['gender'].unique()
```

Out[29]:

```
array(['Male', 'Female'], dtype=object)
```

In [30]:

```
gender_map = {"Female":0,"Male":1}
target_label_map={"Negative":0,"Positive":1}
```

In [31]:

```
# Encode the class using a mapping dictionary
df['gender'] = df['gender'].map(gender_map)
df['class'] = df['class'].map(target_label_map)

```

In [32]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 520 entries, 0 to 519
Data columns (total 17 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   age                 520 non-null    int64
 1   gender              520 non-null    int64
 2   polyuria            520 non-null    int32
 3   polydipsia          520 non-null    int32
 4   sudden_weight_loss  520 non-null    int32
 5   weakness            520 non-null    int32
 6   polyphagia          520 non-null    int32
 7   genital_thrush      520 non-null    int32
 8   visual_blurring     520 non-null    int32
 9   itching             520 non-null    int32
 10  irritability        520 non-null    int32
 11  delayed_healing     520 non-null    int32
 12  partial_paresis     520 non-null    int32
 13  muscle_stiffness    520 non-null    int32
 14  alopecia            520 non-null    int32
 15  obesity             520 non-null    int32
 16  class               520 non-null    int64
dtypes: int32(14), int64(3)
memory usage: 40.8 KB
```

In [33]:

```
 # Recheck Datatypes
df.dtypes
```

Out[33]:

```
age                     int64
gender                  int64
polyuria                int32
polydipsia              int32
sudden_weight_loss      int32
weakness                int32
polyphagia              int32
genital_thrush          int32
visual_blurring         int32
itching                 int32
irritability            int32
delayed_healing         int32
partial_paresis         int32
muscle_stiffness        int32
alopecia                int32
obesity                 int32
class                   int64
dtype: object
```

In [34]:

```
# Recheck using Info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 520 entries, 0 to 519
Data columns (total 17 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   age                 520 non-null    int64
 1   gender              520 non-null    int64
 2   polyuria            520 non-null    int32
 3   polydipsia          520 non-null    int32
 4   sudden_weight_loss  520 non-null    int32
 5   weakness            520 non-null    int32
 6   polyphagia          520 non-null    int32
 7   genital_thrush      520 non-null    int32
 8   visual_blurring     520 non-null    int32
 9   itching             520 non-null    int32
 10  irritability        520 non-null    int32
 11  delayed_healing     520 non-null    int32
 12  partial_paresis     520 non-null    int32
 13  muscle_stiffness    520 non-null    int32
 14  alopecia            520 non-null    int32
 15  obesity             520 non-null    int32
 16  class               520 non-null    int64
dtypes: int32(14), int64(3)
memory usage: 40.8 KB
```

In [35]:

```
# Descriptive Summary
df.describe().T
```

Out[35]:

	count	mean	std	min	25%	50%	75%	max
age	520.0	48.028846	12.151466	16.0	39.0	47.5	57.0	90.0
gender	520.0	0.630769	0.483061	0.0	0.0	1.0	1.0	1.0
polyuria	520.0	0.496154	0.500467	0.0	0.0	0.0	1.0	1.0
polydipsia	520.0	0.448077	0.497776	0.0	0.0	0.0	1.0	1.0
sudden_weight_loss	520.0	0.417308	0.493589	0.0	0.0	0.0	1.0	1.0
weakness	520.0	0.586538	0.492928	0.0	0.0	1.0	1.0	1.0
polyphagia	520.0	0.455769	0.498519	0.0	0.0	0.0	1.0	1.0
genital_thrush	520.0	0.223077	0.416710	0.0	0.0	0.0	0.0	1.0
visual_blurring	520.0	0.448077	0.497776	0.0	0.0	0.0	1.0	1.0
itching	520.0	0.486538	0.500300	0.0	0.0	0.0	1.0	1.0
irritability	520.0	0.242308	0.428892	0.0	0.0	0.0	0.0	1.0
delayed_healing	520.0	0.459615	0.498846	0.0	0.0	0.0	1.0	1.0
partial_paresis	520.0	0.430769	0.495661	0.0	0.0	0.0	1.0	1.0
muscle_stiffness	520.0	0.375000	0.484589	0.0	0.0	0.0	1.0	1.0
alopecia	520.0	0.344231	0.475574	0.0	0.0	0.0	1.0	1.0
obesity	520.0	0.169231	0.375317	0.0	0.0	0.0	0.0	1.0
class	520.0	0.615385	0.486973	0.0	0.0	1.0	1.0	1.0

```
Observation:
1. From the descriptive summary, the minimum age is 16 and the maximum age is 90
2. We will have to get the distribution of data per the age
```

In [36]:

```
# Value Count Per Class
df['class'].value_counts()
```

Out[36]:

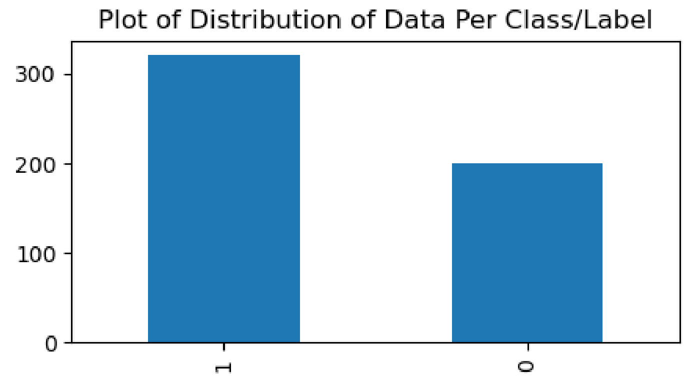
```
1    320
0    200
Name: class, dtype: int64
```

In [37]:

```
 # Plot of Distribution of Data Per Class/Label
```

In [38]:

```
plt.figure(figsize=(5,2.5))
plt.title("Plot of Distribution of Data Per Class/Label")
df['class'].value_counts().plot(kind='bar')
plt.show()
```




```
Observation:
1. Our dataset has
320 datapoints for class 1 Positive) 200 datapoints for class 0( Negative)
2. Balanced dataset from the plot of the value counts
```

In [39]:

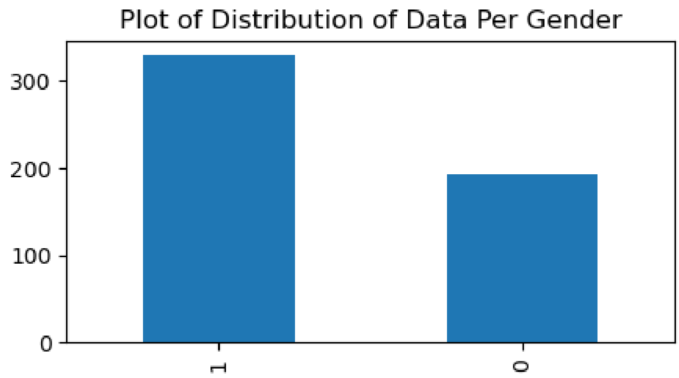
```
# Value Count Of Gender
df['gender'].value_counts()
```

Out[39]:

```
1    328
0    192
Name: gender, dtype: int64
```

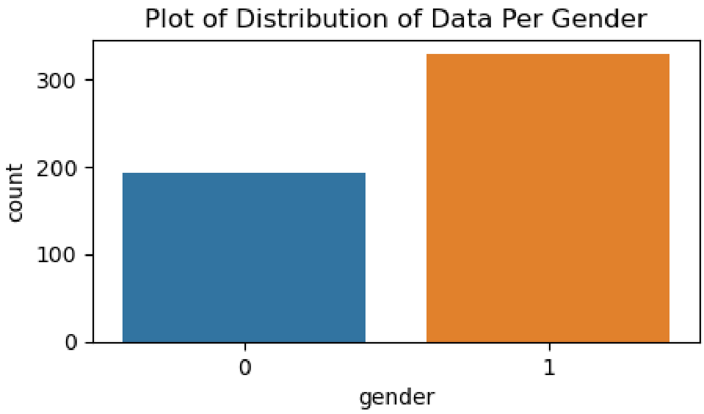
In [40]:

```
# Plot of Distribution of Data Per Gender
plt.figure(figsize=(5,2.5))
plt.title("Plot of Distribution of Data Per Gender")
df['gender'].value_counts().plot(kind='bar')
plt.show()
```



In [41]:

```
# Plot of Distribution of Data Per Gender
plt.figure(figsize=(5,2.5))
plt.title("Plot of Distribution of Data Per Gender")
sns.countplot(x= 'gender', data=df)
plt.show()
```




```
Observation:
1. Our dataset has
328 datapoints for class 1(Males) 192 datapoints for class (Females)
2. There are more males than females
```

In [42]:

```
# Frequency Distribution Table using the Age Range¶
#### Find the minimum and max age
print("Max", df['age'].max())
print("Min", df['age'].min())
```

```
Max 90
Min 16
```

In [43]:

```
labels = ["Less than 10","10-20","20-30","30-40","40-50","50-60","60-70","70-80","80-90"]
bins= [0,10,20,30,40,50,60,70,80,90]
```

In [44]:

```
pd.cut (df['age'],bins=bins, labels=labels)
```

Out[44]:

```
0      30-40
1      50-60
2      40-50
3      40-50
4      50-60
       ...
515    30-40
516    40-50
517    50-60
518    30-40
519    40-50
Name: age, Length: 520, dtype: category
Categories (9, object): ['Less than 10' < '10-20' < '20-30' < '30-40' ... '50-60' < '60-70' < '70-80'
< '80-90']
```

In [45]:

```
fr = pd.cut(df['age'],bins=bins, labels=labels)
```

In [46]:

```
freq_df = df.groupby(fr).size()
```

In [47]:

```
freq_df.head(15)
```

Out[47]:

```
age
Less than 10      0
10-20             1
20-30            44
30-40           123
40-50           145
50-60           127
60-70            66
70-80            10
80-90             4
dtype: int64
```

In [48]:

```
freq_df = freq_df.reset_index(name='count')
```

In [49]:

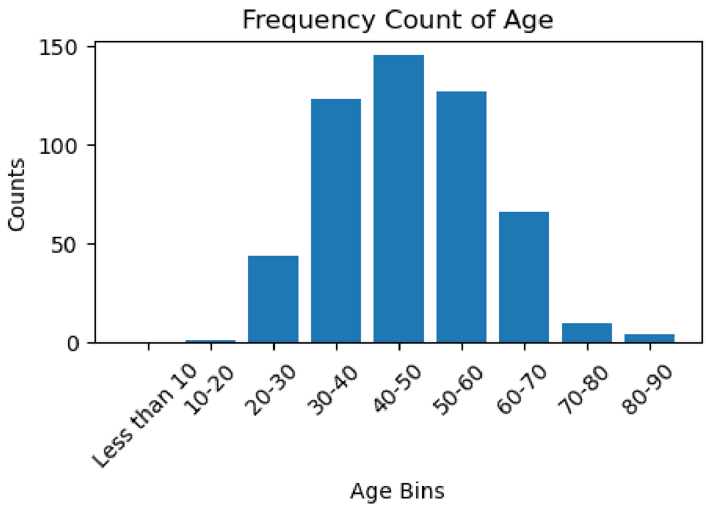
```
freq_df
```

Out[49]:

	age	count
0	Less than 10	0
1	10-20	1
2	20-30	44
3	30-40	123
4	40-50	145
5	50-60	127
6	60-70	66
7	70-80	10
8	80-90	4

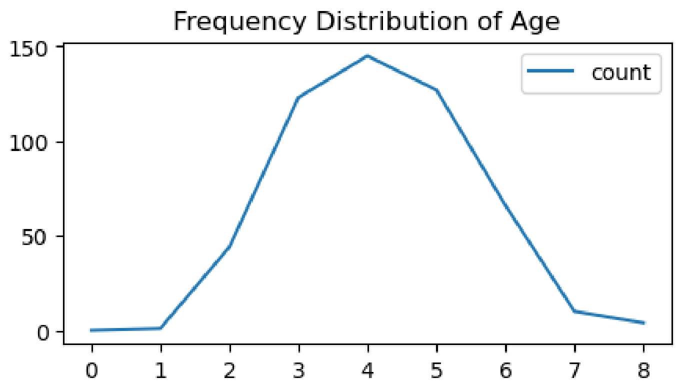
In [50]:

```
# Plot of Distribution of Data Per Gender
plt.figure(figsize=(5,2.5))
plt.bar(freq_df['age'], freq_df['count'])
plt.ylabel('Counts')
plt.xlabel('Age Bins')
plt.xticks(rotation =45)
plt.title('Frequency Count of Age')
plt.show()
```



In [51]:

```
# Plot of Distribution of Data Per Gender
freq_df.plot(kind='line',figsize=(5,2.5))
plt.title("Frequency Distribution of Age")
plt.show()
```



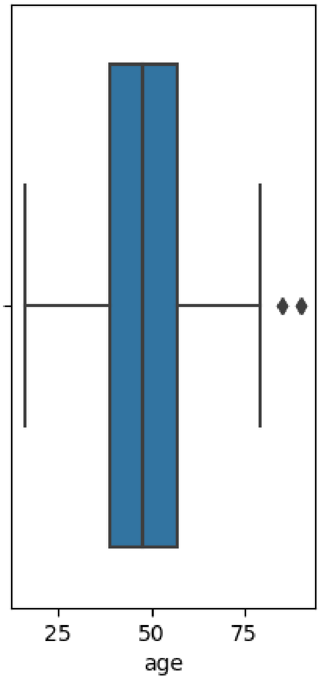

```
Observation:
1. Highest prevalence of Diabetes is from 40-50 followed by 50-60 and 30-40
2. The least is individual under 20, and elderly above 80
```

In [52]:

```
# Find Outliers in Age using BoxPlot
plt.figure(figsize=(2.5,5))
sns.boxplot (df['age'])
```

Out[52]:

```
<AxesSubplot:xlabel='age'>
```



Correlation Analysis of Features in Relation to Target Class (Early Stage Risk)

In [53]:

```
# Method
df.corr()
```

Out[53]:

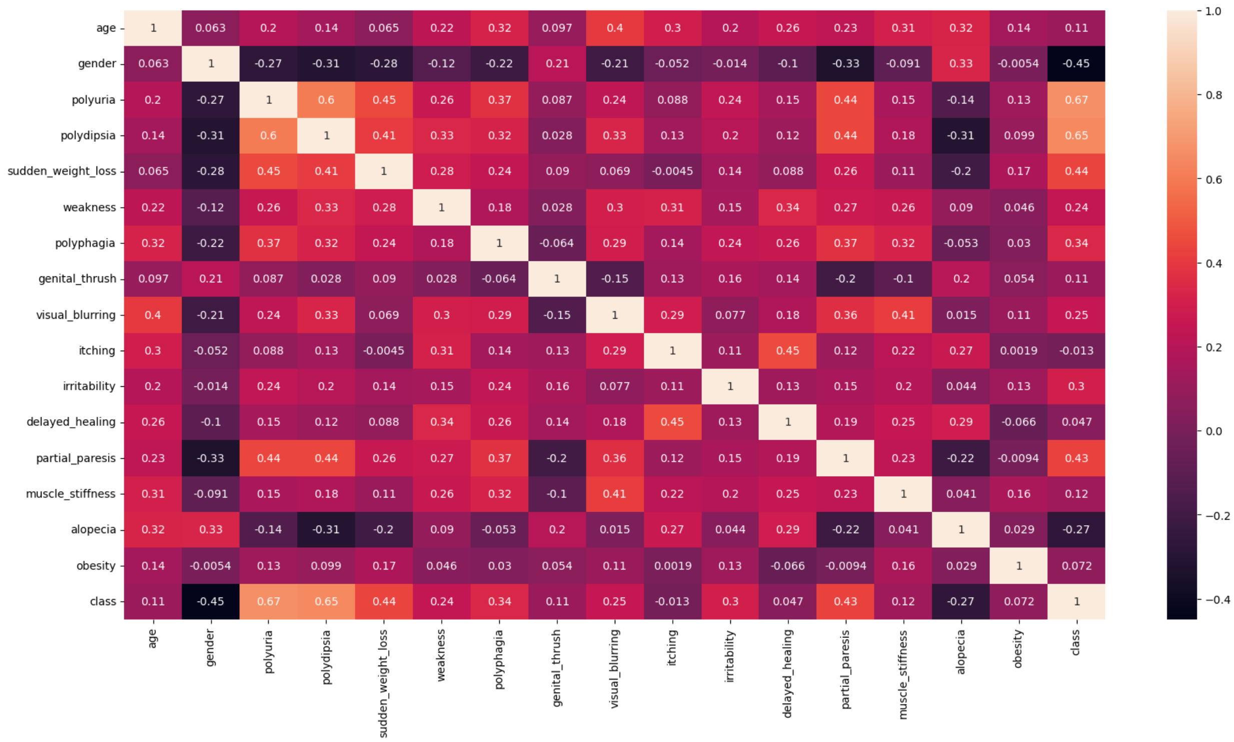
	age	gender	polyuria	polydipsia	sudden_weight_loss	weakness	polyphagia	genital_thrush	visua
age	1.000000	0.062872	0.199781	0.137382	0.064808	0.224596	0.315577	0.096519	
gender	0.062872	1.000000	-0.268894	-0.312262	-0.281840	-0.124490	-0.219968	0.208961	
polyuria	0.199781	-0.268894	1.000000	0.598609	0.447207	0.263000	0.373873	0.087273	
polydipsia	0.137382	-0.312262	0.598609	1.000000	0.405965	0.332453	0.316839	0.028081	
sudden_weight_loss	0.064808	-0.281840	0.447207	0.405965	1.000000	0.282884	0.243511	0.089858	
weakness	0.224596	-0.124490	0.263000	0.332453	0.282884	1.000000	0.180266	0.027780	
polyphagia	0.315577	-0.219968	0.373873	0.316839	0.243511	0.180266	1.000000	-0.063712	
genital_thrush	0.096519	0.208961	0.087273	0.028081	0.089858	0.027780	-0.063712	1.000000	
visual_blurring	0.402729	-0.208092	0.235095	0.331250	0.068754	0.301043	0.293545	-0.148408	
itching	0.296559	-0.052496	0.088289	0.128716	-0.004516	0.309440	0.144390	0.125336	
irritability	0.201625	-0.013735	0.237740	0.203446	0.140340	0.146698	0.239466	0.160551	
delayed_healing	0.257501	-0.101978	0.149873	0.115691	0.088140	0.335507	0.263980	0.136111	
partial_paresis	0.232742	-0.332288	0.441664	0.442249	0.264014	0.272982	0.373569	-0.195612	
muscle_stiffness	0.307703	-0.090542	0.152938	0.180723	0.109756	0.263164	0.320031	-0.100188	
alopecia	0.321691	0.327871	-0.144192	-0.310964	-0.202727	0.090490	-0.053498	0.204847	
obesity	0.140458	-0.005396	0.126567	0.098691	0.169294	0.045665	0.029785	0.053828	
class	0.108679	-0.449233	0.665922	0.648734	0.436568	0.243275	0.342504	0.110288	

In [54]:

```
corr_matrix = df.corr()
```

In [55]:

```
# Plot Correlation with Heatmap
plt.figure(figsize=(20,10))
sns.heatmap(df.corr(), annot=True)
plt.show()
```



In [56]:

```
corr_matrix
```

Out[56]:

	age	gender	polyuria	polydipsia	sudden_weight_loss	weakness	polyphagia	genital_thrush	visu
age	1.000000	0.062872	0.199781	0.137382	0.064808	0.224596	0.315577	0.096519	
gender	0.062872	1.000000	-0.268894	-0.312262	-0.281840	-0.124490	-0.219968	0.208961	
polyuria	0.199781	-0.268894	1.000000	0.598609	0.447207	0.263000	0.373873	0.087273	
polydipsia	0.137382	-0.312262	0.598609	1.000000	0.405965	0.332453	0.316839	0.028081	
sudden_weight_loss	0.064808	-0.281840	0.447207	0.405965	1.000000	0.282884	0.243511	0.089858	
weakness	0.224596	-0.124490	0.263000	0.332453	0.282884	1.000000	0.180266	0.027780	
polyphagia	0.315577	-0.219968	0.373873	0.316839	0.243511	0.180266	1.000000	-0.063712	
genital_thrush	0.096519	0.208961	0.087273	0.028081	0.089858	0.027780	-0.063712	1.000000	
visual_blurring	0.402729	-0.208092	0.235095	0.331250	0.068754	0.301043	0.293545	-0.148408	
itching	0.296559	-0.052496	0.088289	0.128716	-0.004516	0.309440	0.144390	0.125336	
irritability	0.201625	-0.013735	0.237740	0.203446	0.140340	0.146698	0.239466	0.160551	
delayed_healing	0.257501	-0.101978	0.149873	0.115691	0.088140	0.335507	0.263980	0.136111	
partial_paresis	0.232742	-0.332288	0.441664	0.442249	0.264014	0.272982	0.373569	-0.195612	
muscle_stiffness	0.307703	-0.090542	0.152938	0.180723	0.109756	0.263164	0.320031	-0.100188	
alopecia	0.321691	0.327871	-0.144192	-0.310964	-0.202727	0.090490	-0.053498	0.204847	
obesity	0.140458	-0.005396	0.126567	0.098691	0.169294	0.045665	0.029785	0.053828	
class	0.108679	-0.449233	0.665922	0.648734	0.436568	0.243275	0.342504	0.110288	

In [57]:

```
type(corr_matrix)
```

Out[57]:

```
pandas.core.frame.DataFrame
```

In [58]:

```
highest_corr = corr_matrix[corr_matrix >= 0.3]
```

In [59]:

```
highest_corr
```

Out[59]:

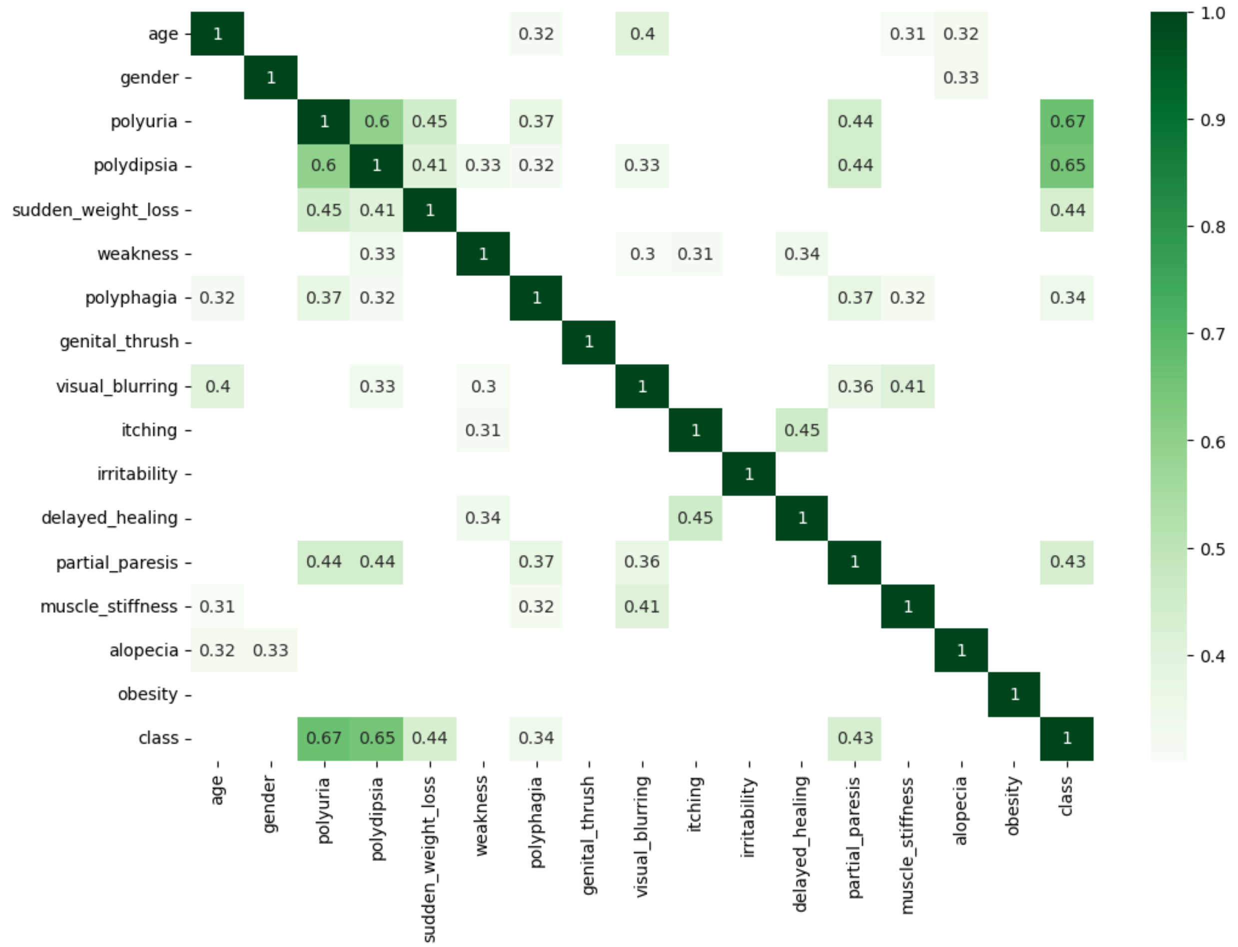
	age	gender	polyuria	polydipsia	sudden_weight_loss	weakness	polyphagia	genital_thrush	visual
age	1.000000	NaN	NaN	NaN	NaN	NaN	0.315577	NaN	
gender	NaN	1.000000	NaN	NaN	NaN	NaN	NaN	NaN	
polyuria	NaN	NaN	1.000000	0.598609	0.447207	NaN	0.373873	NaN	
polydipsia	NaN	NaN	0.598609	1.000000	0.405965	0.332453	0.316839	NaN	
sudden_weight_loss	NaN	NaN	0.447207	0.405965	1.000000	NaN	NaN	NaN	
weakness	NaN	NaN	NaN	0.332453	NaN	1.000000	NaN	NaN	
polyphagia	0.315577	NaN	0.373873	0.316839	NaN	NaN	1.000000	NaN	
genital_thrush	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.0	
visual_blurring	0.402729	NaN	NaN	0.331250	NaN	0.301043	NaN	NaN	
itching	NaN	NaN	NaN	NaN	NaN	0.309440	NaN	NaN	
irritability	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
delayed_healing	NaN	NaN	NaN	NaN	NaN	0.335507	NaN	NaN	
partial_paresis	NaN	NaN	0.441664	0.442249	NaN	NaN	0.373569	NaN	
muscle_stiffness	0.307703	NaN	NaN	NaN	NaN	NaN	0.320031	NaN	
alopecia	0.321691	0.327871	NaN	NaN	NaN	NaN	NaN	NaN	
obesity	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
class	NaN	NaN	0.665922	0.648734	0.436568	NaN	0.342504	NaN	

In [60]:

```
plt.figure(figsize=(12,8))
sns.heatmap(highest_corr, cmap="Greens", annot=True)
```

Out[60]:

<AxesSubplot:>



In [61]:

```
# List Features with the highest Correlation Coefficient
s = corr_matrix.abs()
```

In [62]:

```
s
```

Out[62]:

	age	gender	polyuria	polydipsia	sudden_weight_loss	weakness	polyphagia	genital_thrush	visual_
age	1.000000	0.062872	0.199781	0.137382	0.064808	0.224596	0.315577	0.096519	
gender	0.062872	1.000000	0.268894	0.312262	0.281840	0.124490	0.219968	0.208961	
polyuria	0.199781	0.268894	1.000000	0.598609	0.447207	0.263000	0.373873	0.087273	
polydipsia	0.137382	0.312262	0.598609	1.000000	0.405965	0.332453	0.316839	0.028081	
sudden_weight_loss	0.064808	0.281840	0.447207	0.405965	1.000000	0.282884	0.243511	0.089858	
weakness	0.224596	0.124490	0.263000	0.332453	0.282884	1.000000	0.180266	0.027780	
polyphagia	0.315577	0.219968	0.373873	0.316839	0.243511	0.180266	1.000000	0.063712	
genital_thrush	0.096519	0.208961	0.087273	0.028081	0.089858	0.027780	0.063712	1.000000	
visual_blurring	0.402729	0.208092	0.235095	0.331250	0.068754	0.301043	0.293545	0.148408	
itching	0.296559	0.052496	0.088289	0.128716	0.004516	0.309440	0.144390	0.125336	
irritability	0.201625	0.013735	0.237740	0.203446	0.140340	0.146698	0.239466	0.160551	
delayed_healing	0.257501	0.101978	0.149873	0.115691	0.088140	0.335507	0.263980	0.136111	
partial_paresis	0.232742	0.332288	0.441664	0.442249	0.264014	0.272982	0.373569	0.195612	
muscle_stiffness	0.307703	0.090542	0.152938	0.180723	0.109756	0.263164	0.320031	0.100188	
alopecia	0.321691	0.327871	0.144192	0.310964	0.202727	0.090490	0.053498	0.204847	
obesity	0.140458	0.005396	0.126567	0.098691	0.169294	0.045665	0.029785	0.053828	
class	0.108679	0.449233	0.665922	0.648734	0.436568	0.243275	0.342504	0.110288	

In [63]:

```
s.unstack()
```

Out[63]:

```
age    age                   1.000000
       gender                0.062872
       polyuria              0.199781
       polydipsia            0.137382
       sudden_weight_loss    0.064808
                               ...
class  partial_paresis       0.432288
       muscle_stiffness      0.122474
       alopecia              0.267512
       obesity               0.072173
       class                 1.000000
Length: 289, dtype: float64
```

In [64]:

```
 s = corr_matrix.abs().unstack()
```

In [65]:

```
top_features_per_correlation =s.sort_values(kind="quicksort")
```

In [66]:

```
top_features_per_correlation
```

Out[66]:

```
obesity            itching             0.001894
itching            obesity             0.001894
                   sudden_weight_loss  0.004516
sudden_weight_loss itching             0.004516
obesity            gender              0.005396
                                         ...
polydipsia         polydipsia          1.000000
polyuria           polyuria            1.000000
gender             gender              1.000000
obesity            obesity             1.000000
class              class               1.000000
Length: 289, dtype: float64
```

Feature Engineering and Selection

1. A feature is an attribute or property shared by all of the independent units on which analysis or prediction is to be done.
2. Feature engineering is the process of using domain knowledge to extract features from raw data via data mining techniques. These features can be used to improve the performance of machine learning algorithms.
3. We will be using feature selection techniques to find the most informative features for our model.

In [67]:

```
from sklearn.feature_selection import SelectKBest, chi2, RFE
from sklearn.ensemble import ExtraTreesClassifier
```

In [68]:

```
 # Features and Labels
# Which columns are for features and for labels
X = df[['age', 'gender', 'polyuria', 'polydipsia', 'sudden_weight_loss',
'weakness', 'polyphagia', 'genital_thrush', 'visual_blurring', 'itching', 'irritability', 'delayed_healing','par
y = df['class']

```

In [69]:

```
# Find the best features using Selectkbest 2
skb = SelectKBest (score_func=chi2, k=10)
```

In [70]:

```
best_feature_fit = skb.fit(X,y)
```

In [71]:

```
best_feature_fit.scores_
```

Out[71]:

```
array([1.88457668e+01, 3.87476372e+01, 1.16184593e+02, 1.20785515e+02,
       5.77493088e+01, 1.27242623e+01, 3.31984177e+01, 4.91400862e+00,
       1.81245708e+01, 4.78260870e-02, 3.53341270e+01, 6.20188285e-01,
       5.53142857e+01, 4.87500000e+00, 2.44027933e+01, 2.25028409e+00])
```

In [72]:

```
# Mapping to Feature Name
feature_scores = pd.DataFrame(best_feature_fit.scores_, columns=['Feature_Score'])
```

In [73]:

```
feature_scores
```

Out[73]:

	Feature_Score
0	18.845767
1	38.747637
2	116.184593
3	120.785515
4	57.749309
5	12.724262
6	33.198418
7	4.914009
8	18.124571
9	0.047826
10	35.334127
11	0.620188
12	55.314286
13	4.875000
14	24.402793
15	2.250284

In [74]:

```
X.columns
```

Out[74]:

```
Index(['age', 'gender', 'polyuria', 'polydipsia', 'sudden_weight_loss',
       'weakness', 'polyphagia', 'genital_thrush', 'visual_blurring',
       'itching', 'irritability', 'delayed_healing', 'partial_paresis',
       'muscle_stiffness', 'alopecia', 'obesity'],
      dtype='object')
```

In [75]:

```
feature_column_names = pd.DataFrame(X.columns,columns=['Feature_names'])
```

In [76]:

```
feature_column_names
```

Out[76]:

	Feature_names
0	age
1	gender
2	polyuria
3	polydipsia
4	sudden_weight_loss
5	weakness
6	polyphagia
7	genital_thrush
8	visual_blurring
9	itching
10	irritability
11	delayed_healing
12	partial_paresis
13	muscle_stiffness
14	alopecia
15	obesity

In [77]:

```
best_feat_df = pd.concat([feature_scores, feature_column_names], axis=1)
```

In [78]:

```
best_feat_df
```

Out[78]:

	Feature_Score	Feature_names
0	18.845767	age
1	38.747637	gender
2	116.184593	polyuria
3	120.785515	polydipsia
4	57.749309	sudden_weight_loss
5	12.724262	weakness
6	33.198418	polyphagia
7	4.914009	genital_thrush
8	18.124571	visual_blurring
9	0.047826	itching
10	35.334127	irritability
11	0.620188	delayed_healing
12	55.314286	partial_paresis
13	4.875000	muscle_stiffness
14	24.402793	alopecia
15	2.250284	obesity

In [79]:

```
best_feat_df.sort_values (by=['Feature_Score'], ascending=False)
```

Out[79]:

	Feature_Score	Feature_names
3	120.785515	polydipsia
2	116.184593	polyuria
4	57.749309	sudden_weight_loss
12	55.314286	partial_paresis
1	38.747637	gender
10	35.334127	irritability
6	33.198418	polyphagia
14	24.402793	alopecia
0	18.845767	age
8	18.124571	visual_blurring
5	12.724262	weakness
7	4.914009	genital_thrush
13	4.875000	muscle_stiffness
15	2.250284	obesity
11	0.620188	delayed_healing
9	0.047826	itching

In [80]:

```
best_feat_df.nlargest (10, 'Feature_Score')
```

Out[80]:

	Feature_Score	Feature_names
3	120.785515	polydipsia
2	116.184593	polyuria
4	57.749309	sudden_weight_loss
12	55.314286	partial_paresis
1	38.747637	gender
10	35.334127	irritability
6	33.198418	polyphagia
14	24.402793	alopecia
0	18.845767	age
8	18.124571	visual_blurring

In [81]:

```
# List Columns/Features we will be using
best_feat_df.nlargest (10, 'Feature_Score')['Feature_names'].unique()
```

Out[81]:

```
array(['polydipsia', 'polyuria', 'sudden_weight_loss', 'partial_paresis',
       'gender', 'irritability', 'polyphagia', 'alopecia', 'age',
       'visual_blurring'], dtype=object)
```

```
Observation:
1. From our analysis, polydipsia, polyuria, sudden weight loss and partial paresis plays an important role in making our prediction
2. This confirms an already established fact for signs of diabetes ie, polydipsia, polyuria and polyphagia.
```

Which of these features are important using ExtraTrees Classifier

In [82]:

```
et_clf = ExtraTreesClassifier()
```

In [83]:

```
et_clf.fit(X,y)
```

Out[83]:

```
ExtraTreesClassifier()
```

In [84]:

```
 # Print Important
print(et_clf.feature_importances_)
```

```
[0.04851932 0.10575599 0.25226137 0.16899971 0.06486471 0.02162635
 0.02850983 0.02378278 0.03450691 0.03567815 0.04375055 0.04033182
 0.04737593 0.02529199 0.03686605 0.02187853]
```

In [85]:

```
# Mapping to Feature Name
feature_importance_df = pd.Series(et_clf.feature_importances_,index=X.columns)
```

In [86]:

```
feature_importance_df.head()
```

Out[86]:

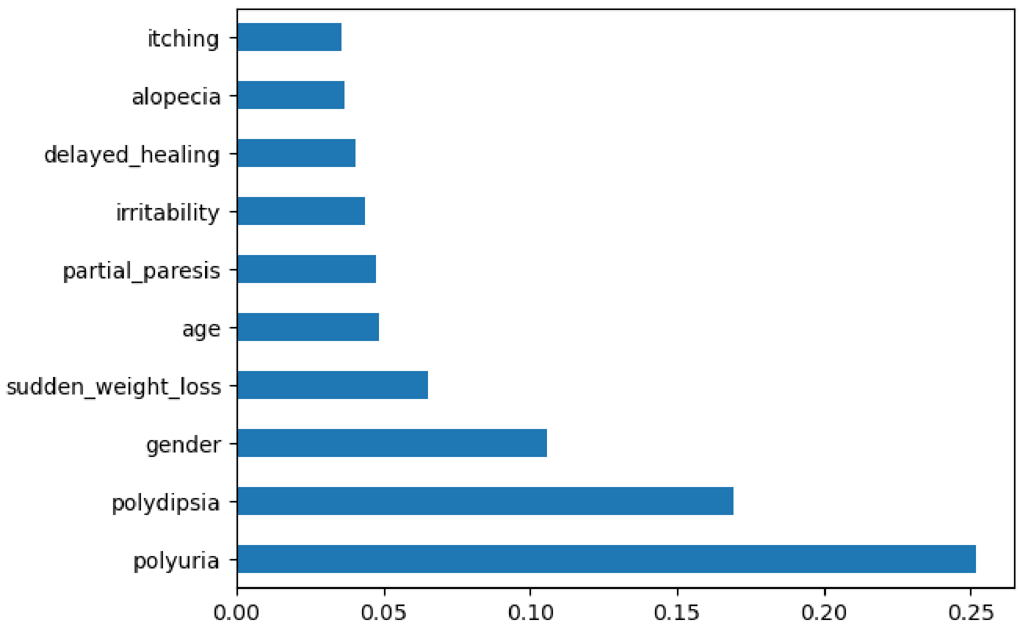
```
age                   0.048519
gender                0.105756
polyuria              0.252261
polydipsia            0.169000
sudden_weight_loss    0.064865
dtype: float64
```

In [87]:

```
feature_importance_df.nlargest (10).plot(kind='barh')
```

Out[87]:

```
<AxesSubplot:>
```




```
Obsevations
```

```
3 1. Using ExtraTreeClassifier Algorithm we found out similar result with
the SelectKBest
4 2. Polyuria, polydipsia, gender and sudden weight loss, age and partial
paresis are the most important.
5 3. Almost like the previous except that the order of gender and sudden
weight loss was changed
6 4. However since this is a health issue we will be using all the features
as there can be diverse scenario due to
7 different life style and physiology of individuals

```

Machine Learning Model Development

In [88]:

```python
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
```

In [89]:

```python
# Split Dataset into 2
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.33, random_state=42)
```

In [90]:

```python
x_test.shape
```

Out[90]:

```
(172, 16)
```

In [91]:

```python
 # Shape of Dataset
```

In [92]:

```python
print("original data", df.shape)
print("training data",x_train.shape)
print("testing data",y_test.shape)
```

```
original data (520, 17)
training data (348, 16)
testing data (172,)
```

In [93]:

```python
 # Using LogisticRegression Estimator to Build A Model
lr_model = LogisticRegression()
```

In [94]:

```python
lr_model.fit(x_train,y_train)
```

Out[94]:

```
LogisticRegression()
```

In [95]:

```python
# Check Model Accuracy
# Method 1
lr_model.score (x_train,y_train)

```

Out[95]:

```
0.9310344827586207
```

In [96]:

```
y_pred=lr_model.predict(x_test)
```

In [97]:

```
 # Using Accuracy Score to check for accuracy by comparing with the predict
print (f"Accuracy of LR Model: {accuracy_score(y_test,y_pred)}")

```

```
Accuracy of LR Model: 0.936046511627907
```

In [98]:

```
# Using Classification Report
from sklearn.metrics import classification_report

```

In [99]:

```
target_names= ["Negative(0)-No_Sugar", "Positive(1)-Sugar"]
```

In [100]:

```
 # Classification Report
```

In [101]:

```
print(classification_report(y_test,y_pred, target_names=target_names))
```

```
                      precision    recall  f1-score   support

Negative(0)-No_Sugar       0.90      0.92      0.91        61
   Positive(1)-Sugar       0.95      0.95      0.95       111

            accuracy                           0.94       172
           macro avg       0.93      0.93      0.93       172
        weighted avg       0.94      0.94      0.94       172
```

In [102]:

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,y_pred)
```

Out[102]:

```
array([[ 56,   5],
       [  6, 105]], dtype=int64)
```

In [103]:

```
cm=confusion_matrix(y_test,y_pred)
```

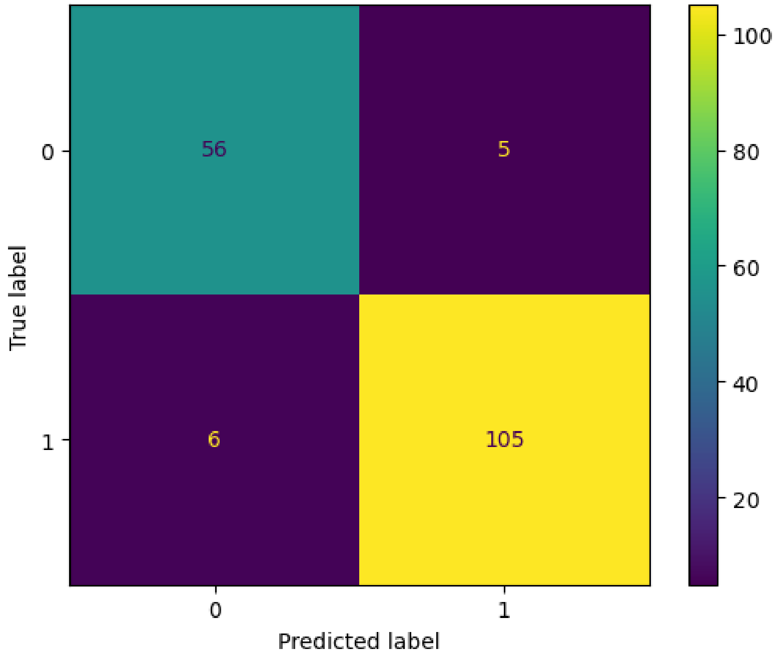
In [104]:

```
from sklearn.metrics import ConfusionMatrixDisplay
```

In [105]:

```
disp = ConfusionMatrixDisplay (confusion_matrix=cm,display_labels=lr_model.classes_)

disp.plot()
plt.show()
```



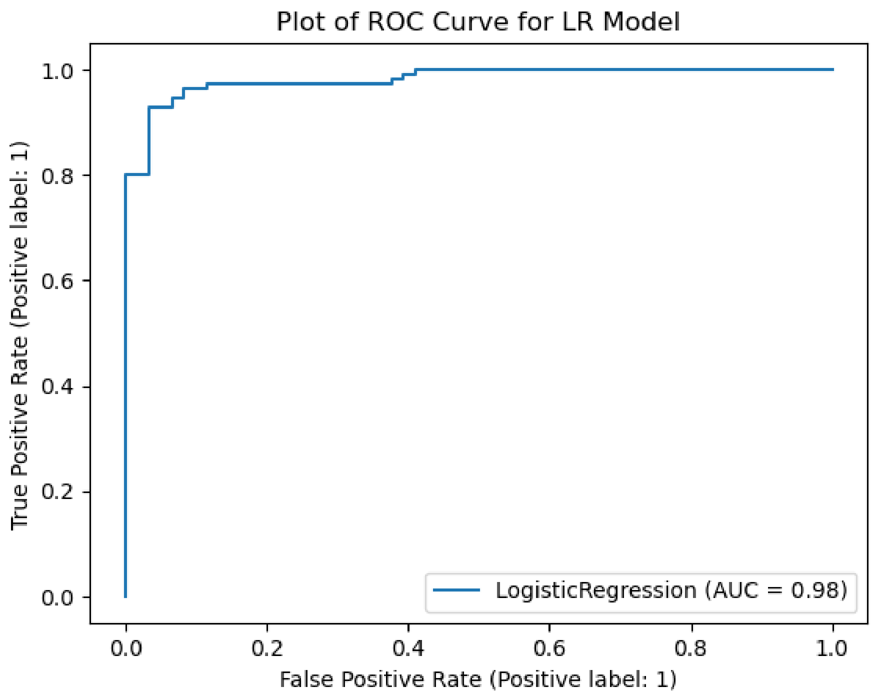
ROC Curve

In [106]:

```
from sklearn.metrics import roc_auc_score, RocCurveDisplay
```

In [107]:

```
#ROC Curve
RocCurveDisplay.from_estimator(lr_model, x_test, y_test)
plt.title("Plot of ROC Curve for LR Model")
plt.show()
```

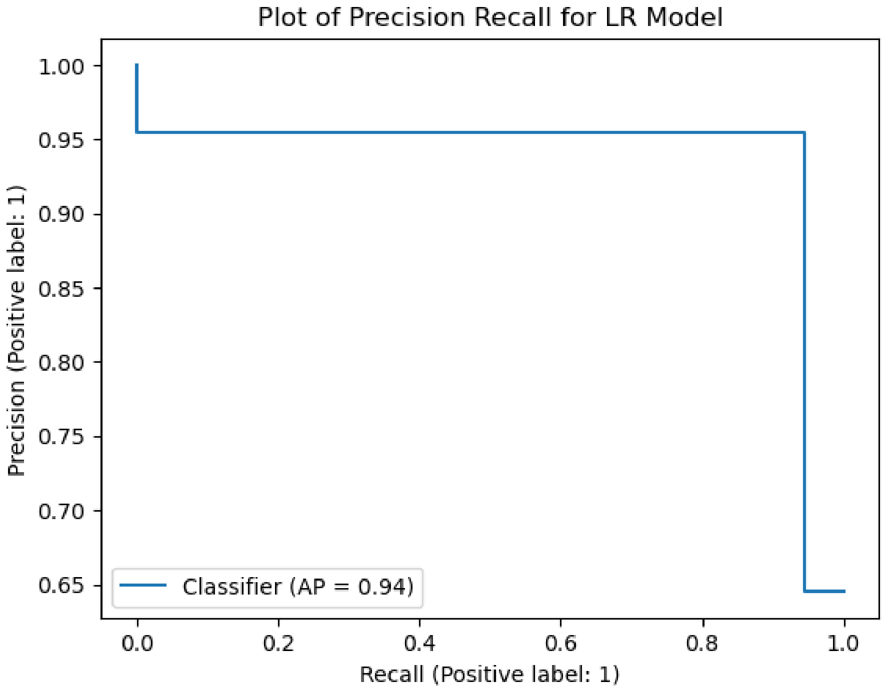


In [108]:

```
from sklearn.metrics import PrecisionRecallDisplay
```

In [109]:

```
PrecisionRecallDisplay.from_predictions (y_test,y_pred)
plt.title("Plot of Precision Recall for LR Model")
plt.show()
```



In [110]:

```
from sklearn.model_selection import cross_val_score
def cross_validate_model(model_estimator,X,y,cv):
    """Evaluate Model using cross validation of KFolds"""
    scores = cross_val_score (model_estimator, X, y, scoring='accuracy', cv=10)
    result = "Mean Accuracy: {} Standard_Deviation : {}".format(np.mean(scores), np.std(scores))
    return result
```

In [111]:

```
 # Cross Validaion For LR
cv_scores_for_lr_model = cross_validate_model(LogisticRegression(),X,y,5)

```

In [112]:

```
cv_scores_for_lr_model
```

Out[112]:

```
'Mean Accuracy: 0.9307692307692307 Standard_Deviation : 0.05029498781008469'
```

Saving model

In [113]:

```
# Using Joblib
import joblib
print("Joblib",joblib.__version__)

```

```
Joblib 1.1.0
```

In [114]:

```
 # Save LR Model
model_file = open("log_regression_05_20_2023.pk1","wb")
joblib.dump(lr_model,model_file)
model_file.close()
```

In [ ]:

```

```

In [ ]:

```

```

Decision Tree

In [157]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [158]:

```
model = DecisionTreeClassifier()
```

In [159]:

```
model.fit(x_train,y_train)
```

Out[159]:

```
DecisionTreeClassifier()
```

In [160]:

```
model.score(x_train,y_train)
```

Out[160]:

```
1.0
```

In [161]:

```
y_predict = model.predict(x_test)
```

In [162]:

```
from sklearn.metrics import accuracy_score
```

In [163]:

```
accuracy_score(y_predict,y_test)
```

Out[163]:

```
0.9651162790697675
```

In [164]:

```
model_1 = DecisionTreeClassifier(criterion='entropy',max_depth=8,min_samples_split=10,splitter='random')
```

In [165]:

```
model_1.fit(x_train,y_train)
```

Out[165]:

```
DecisionTreeClassifier(criterion='entropy', max_depth=8, min_samples_split=10,
                       splitter='random')
```

In [166]:

```
model_1.score(x_train,y_train)
```

Out[166]:

```
0.9741379310344828
```

In [167]:

```
model_1.fit(x_train,y_train)
```

Out[167]:

```
DecisionTreeClassifier(criterion='entropy', max_depth=8, min_samples_split=10,
                       splitter='random')
```

In [168]:

```
from sklearn.metrics import accuracy_score
```

In [169]:

```
accuracy_score(y_predict,y_test)
```

Out[169]:

```
0.9651162790697675
```

In [170]:

```
# GridSearchCV
```

In [171]:

```
from sklearn.model_selection import GridSearchCV
# grid_search = GridSearchCV(estimator=model,param_grid=grid_param,cv=5)
```

In [172]:

```
grid_param = {
'criterion':['gini','entropy'],
'max_depth':range(2,32,1),
'min_samples_split': range(2,10,1),
'min_samples_leaf' : range(1,10,1),
'splitter': ['best', 'random']
}
```

In [173]:

```
grid_search = GridSearchCV(estimator=model,param_grid=grid_param,cv=5)
```

In [174]:

```
grid_search.fit(x_train,y_train)
```

Out[174]:

```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(2, 32),
                         'min_samples_leaf': range(1, 10),
                         'min_samples_split': range(2, 10),
                         'splitter': ['best', 'random']})
```

In [175]:

```
grid_search.get_params
```

Out[175]:

```
<bound method BaseEstimator.get_params of GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(2, 32),
                         'min_samples_leaf': range(1, 10),
                         'min_samples_split': range(2, 10),
                         'splitter': ['best', 'random']})>
```

In [176]:

```
model_3 = DecisionTreeClassifier(criterion= 'gini',max_depth= 5,min_samples_split=6,splitter='random')
```

In [223]:

```
model_3.fit(x_train,y_train)
```

Out[223]:

```
DecisionTreeClassifier(max_depth=5, min_samples_split=6, splitter='random')
```

In [224]:

```
model_3.score(x_train,y_train)
```

Out[224]:

```
0.9626436781609196
```

In [226]:

```
y_predict = model_3.predict(x_test)
```

In [227]:

```
from sklearn.metrics import accuracy_score
```

In [228]:

```
accuracy_score(y_predict,y_test)
```

Out[228]:

```
0.9593023255813954
```

In [ ]:

```

```

In [ ]:

```

```

Random Forest

In [115]:

```
from sklearn.model_selection import train_test_split
```

In [116]:

```
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.33, random_state=42)
```

In [129]:

```
y.value_counts()
```

Out[129]:

```
1    320
0    200
Name: class, dtype: int64
```

In [124]:

```
from sklearn.ensemble import RandomForestClassifier
rf_classifier=RandomForestClassifier(n_estimators=20)
```

In [125]:

```
rf_classifier.fit(x_train,y_train)
```

Out[125]:

```
RandomForestClassifier(n_estimators=20)
```

In [126]:

```
rf_classifier.score(x_train,y_train)
```

Out[126]:

```
1.0
```

In [127]:

```
y_predict = rf_classifier.predict(x_test)
```

In [128]:

```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
print(confusion_matrix(y_test,y_predict))
print(classification_report(y_test,y_predict))
print(accuracy_score(y_test,y_predict))
```

```
[[ 60   1]
 [  3 108]]
              precision    recall  f1-score   support

           0       0.95      0.98      0.97        61
           1       0.99      0.97      0.98       111

    accuracy                           0.98       172
   macro avg       0.97      0.98      0.97       172
weighted avg       0.98      0.98      0.98       172

0.9767441860465116
```

In [130]:

```
##manual Hyperparameter tuning
```

In [131]:

```
model=RandomForestClassifier(n_estimators=300,criterion='entropy',max_features='sqrt',min_samples_leaf=10,random_
model.fit(x_train,y_train)
y_predict = rf_classifier.predict(x_test)
print(confusion_matrix(y_test,y_predict))
print(classification_report(y_test,y_predict))
print(accuracy_score(y_test,y_predict))
```

```
[[ 60   1]
 [  3 108]]
              precision    recall  f1-score   support

           0       0.95      0.98      0.97        61
           1       0.99      0.97      0.98       111

    accuracy                           0.98       172
   macro avg       0.97      0.98      0.97       172
weighted avg       0.98      0.98      0.98       172

0.9767441860465116
```

Random Search CV

In [140]:

```
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt','log2']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 800,10)]
# Minimum number of samples required to split a node
min_samples_split = [1,2,3,5,6,7]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2,3, 4,6,8]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
              'criterion':['entropy','gini']}
print(random_grid)
```

```
{'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000], 'max_features': ['auto', 'sqrt',
'log2'], 'max_depth': [10, 97, 185, 273, 361, 448, 536, 624, 712, 800], 'min_samples_split': [1, 2, 3,
5, 6, 7], 'min_samples_leaf': [1, 2, 3, 4, 6, 8], 'criterion': ['entropy', 'gini']}
```

In [141]:

```
rf=RandomForestClassifier()
rf_randomcv=RandomizedSearchCV(estimator=rf,param_distributions=random_grid,n_iter=100,cv=3,verbose=2,
                               random_state=100,n_jobs=-1)
### fit the randomized model
rf_randomcv.fit(x_train,y_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

Out[141]:

```
RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(), n_iter=100,
                   n_jobs=-1,
                   param_distributions={'criterion': ['entropy', 'gini'],
                                        'max_depth': [10, 97, 185, 273, 361,
                                                      448, 536, 624, 712, 800],
                                        'max_features': ['auto', 'sqrt',
                                                         'log2'],
                                        'min_samples_leaf': [1, 2, 3, 4, 6, 8],
                                        'min_samples_split': [1, 2, 3, 5, 6, 7],
                                        'n_estimators': [100, 200, 300, 400,
                                                         500, 600, 700, 800,
                                                         900, 1000]},
                   random_state=100, verbose=2)
```

In [142]:

```
## to find best parameteres
rf_randomcv.best_params_
```

Out[142]:

```
{'n_estimators': 400,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': 800,
 'criterion': 'gini'}
```

In [143]:

```
rf_randomcv
```

Out[143]:

```
RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(), n_iter=100,
                   n_jobs=-1,
                   param_distributions={'criterion': ['entropy', 'gini'],
                                        'max_depth': [10, 97, 185, 273, 361,
                                                      448, 536, 624, 712, 800],
                                        'max_features': ['auto', 'sqrt',
                                                         'log2'],
                                        'min_samples_leaf': [1, 2, 3, 4, 6, 8],
                                        'min_samples_split': [1, 2, 3, 5, 6, 7],
                                        'n_estimators': [100, 200, 300, 400,
                                                         500, 600, 700, 800,
                                                         900, 1000]},
                   random_state=100, verbose=2)
```

In [144]:

```
best_random_grid=rf_randomcv.best_estimator_
```

In [146]:

```
from sklearn.metrics import accuracy_score
y_pred=best_random_grid.predict(x_test)
print(confusion_matrix(y_test,y_pred))
print("Accuracy Score {}".format(accuracy_score(y_test,y_pred)))
print("Classification report: {}".format(classification_report(y_test,y_pred)))
```

```
[[ 61   0]
 [  2 109]]
Accuracy Score 0.9883720930232558
Classification report:               precision    recall  f1-score   support

           0       0.97      1.00      0.98        61
           1       1.00      0.98      0.99       111

    accuracy                           0.99       172
   macro avg       0.98      0.99      0.99       172
weighted avg       0.99      0.99      0.99       172
```

Grid Search CV

In [147]:

```
rf_randomcv.best_params_
```

Out[147]:

```
{'n_estimators': 400,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': 800,
 'criterion': 'gini'}
```

In [148]:

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'criterion': [rf_randomcv.best_params_['criterion']],
    'max_depth': [rf_randomcv.best_params_['max_depth']],
    'max_features': [rf_randomcv.best_params_['max_features']],
    'min_samples_leaf': [rf_randomcv.best_params_['min_samples_leaf'],
                         rf_randomcv.best_params_['min_samples_leaf']+2,
                         rf_randomcv.best_params_['min_samples_leaf'] + 4],
    'min_samples_split': [rf_randomcv.best_params_['min_samples_split'] - 2,
                          rf_randomcv.best_params_['min_samples_split'] - 1,
                          rf_randomcv.best_params_['min_samples_split'],
                          rf_randomcv.best_params_['min_samples_split'] +1,
                          rf_randomcv.best_params_['min_samples_split'] + 2],
    'n_estimators': [rf_randomcv.best_params_['n_estimators'] - 200, rf_randomcv.best_params_['n_estimators'] - 
                     rf_randomcv.best_params_['n_estimators'],
                     rf_randomcv.best_params_['n_estimators'] + 300, rf_randomcv.best_params_['n_estimators'] + 
}

print(param_grid)
```

```
{'criterion': ['gini'], 'max_depth': [800], 'max_features': ['auto'], 'min_samples_leaf': [1, 3, 5],
'min_samples_split': [0, 1, 2, 3, 4], 'n_estimators': [200, 100, 400, 700, 800]}
```

In [151]:

```
### Fit the grid_search to the data
rf=RandomForestClassifier()
grid_search=GridSearchCV(estimator=rf,param_grid=param_grid,cv=5,n_jobs=-1,verbose=2)
grid_search.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 75 candidates, totalling 375 fits
```

Out[151]:

```
GridSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini'], 'max_depth': [800],
                         'max_features': ['auto'],
                         'min_samples_leaf': [1, 3, 5],
                         'min_samples_split': [0, 1, 2, 3, 4],
                         'n_estimators': [200, 100, 400, 700, 800]},
             verbose=2)
```

In [152]:

```
### to find best parameters
grid_search.best_estimator_
```

Out[152]:

```
RandomForestClassifier(max_depth=800)
```

In [153]:

```
best_grid=grid_search.best_estimator_
```

In [154]:

```
best_grid
```

Out[154]:

```
RandomForestClassifier(max_depth=800)
```

In [156]:

```
### Confusion matrix
y_pred=best_grid.predict(x_test)
print(confusion_matrix(y_test,y_pred))
print("Accuracy Score {}".format(accuracy_score(y_test,y_pred)))
print("Classification report: {}".format(classification_report(y_test,y_pred)))
```

```
[[ 61   0]
 [  2 109]]
Accuracy Score 0.9883720930232558
Classification report:               precision    recall  f1-score   support

           0       0.97      1.00      0.98        61
           1       1.00      0.98      0.99       111

    accuracy                           0.99       172
   macro avg       0.98      0.99      0.99       172
weighted avg       0.99      0.99      0.99       172
```

```
Observation:
    Randomforest gave 98 percentage accuracy
```

In [ ]:

```

```