

## Importing All The Useful Libraries

```
import numpy as np
import pandas as pd
import seaborn as srn
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
import pydotplus
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
import plotly.graph_objects as go
from sklearn import svm
```


Using Pandas Library to read the CSV File. We have used 4 functions of the dataframe object:

- 1) Head Function: Shows the first 5 object values of the dataframe
- 2) Tail Function: Shows the last 5 object values of the dataframe
- 3) Describe Function: To give different statistics of the numerical data
- 4) Dtypes function: To show the datatype of different columns of the dataframe

```
df=pd.read_csv("Diabetestype.csv")
df.head()
```

	Age	BS	Fast	BS	pp	Plasma R	Plasma F	HbA1c	Type	Class
0	50		6.8	8.8		11.2	7.2	62	Type1	1
1	31		5.2	6.8		10.9	4.2	33	Normal	0
2	32		6.8	8.8		11.2	7.2	62	Type1	1
3	21		5.7	5.8		10.7	4.8	49	Normal	0
4	33		6.8	8.8		11.2	7.2	62	Type1	1

```
df.tail()
```

	Age	BS Fast	BS pp	Plasma R	Plasma F	HbA1c	Type	Class	
<b>1004</b>	37	0.0	5.6	10.2	5.4	32	Normal	0	
<b>1005</b>	23	6.3	4.2	12.2	7.8	57	Type2	1	
<b>1006</b>	37	32.0	7.4	8.7	5.6	41	Normal	0	
<b>1007</b>	46	19.0	6.3	7.9	3.9	40	Normal	0	

```
df.describe()
```

	Age	BS Fast	BS pp	Plasma R	Plasma F	HbA1c	
<b>count</b>	1009.000000	1009.000000	951.000000	1009.000000	929.000000	1009.000000	1009
<b>mean</b>	33.398414	12.571853	6.646583	10.728741	6.134553	43.481665	0
<b>std</b>	11.633364	12.334019	1.208516	1.436979	1.626551	12.067515	0
<b>min</b>	21.000000	0.000000	4.200000	7.900000	3.900000	28.000000	0
<b>25%</b>	24.000000	5.600000	5.800000	10.200000	4.800000	33.000000	0
<b>50%</b>	29.000000	6.700000	6.800000	10.900000	5.600000	40.000000	0
<b>75%</b>	41.000000	20.000000	7.700000	11.400000	7.800000	53.000000	1
<b>max</b>	81.000000	54.000000	8.800000	13.100000	9.100000	69.000000	1

```
df.dtypes
```

```
Age          int64
BS Fast      float64
BS pp        float64
Plasma R     float64
Plasma F     float64
HbA1c        int64
Type         object
Class        int64
dtype: object
```

## Preprocessing Stage

### First Step:

Our data is dirty because both Type 1 and Type 2 diabetes are represented as 1 in the column Class. So we will differentiate between them by representing Type 1 diabetes as 1 and Type 2 diabetes as 2

```
diatypes=df['Type']
dianum=df['Class']
diatypes
dianum
```

```

0      1
1      0
2      1
3      0
4      1
..
1004    0
1005    1
1006    0
1007    0
1008    0
Name: Class, Length: 1009, dtype: int64

```

```

for i in range(len(diatypes)):
    if(diatypes[i]=="Normal"):
        dianum[i]=0
    else:
        if(diatypes[i]=="Type1"):
            dianum[i]=1
        else:
            dianum[i]=2
dianum

```

/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:6: SettingWithCopyWarning

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: <https://pandas.pydata.org/pandas-docs/stable/10min.html#copy-on-write>

/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:3: SettingWithCopyWarning

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: <https://pandas.pydata.org/pandas-docs/stable/10min.html#copy-on-write>

/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:8: SettingWithCopyWarning

A value is trying to be set on a copy of a slice from a DataFrame


See the caveats in the documentation: <https://pandas.pydata.org/pandas-docs/stable/10min.html#copy-on-write>

```

0      1
1      0
2      1
3      0
4      1
..
1004    0
1005    2
1006    0
1007    0
1008    0
Name: Class, Length: 1009, dtype: int64

```

```
df['Class']=dianum
df
```



	Age	BS Fast	BS pp	Plasma R	Plasma F	HbA1c	Type	Class
<b>0</b>	50	6.8	8.8	11.2	7.2	62	Type1	1
<b>1</b>	31	5.2	6.8	10.9	4.2	33	Normal	0
<b>2</b>	32	6.8	8.8	11.2	7.2	62	Type1	1
<b>3</b>	21	5.7	5.8	10.7	4.8	49	Normal	0
<b>4</b>	33	6.8	8.8	11.2	7.2	62	Type1	1
...	...	...	...	...	...	...	...	...
<b>1004</b>	37	0.0	5.6	10.2	5.4	32	Normal	0
<b>1005</b>	23	6.3	4.2	12.2	7.8	57	Type2	2
<b>1006</b>	37	32.0	7.4	8.7	5.6	41	Normal	0
<b>1007</b>	46	19.0	6.3	7.9	3.9	40	Normal	0
<b>1008</b>	25	16.0	6.3	7.9	3.9	40	Normal	0

1009 rows × 8 columns


### Second Step:

As we can see from data description that BS pp and Plasma F columns have some values missing. We will use Mean to predict the missing values.

```
dfcopy=df
dfcopy['BS pp'].fillna(dfcopy['BS pp'].mean(),inplace=True)


dfcopy['Plasma F'].fillna(dfcopy['Plasma F'].mean(),inplace=True)

df=dfcopy
df
```



	Age	BS Fast	BS pp	Plasma R	Plasma F	HbA1c	Type	Class
0	50	6.8	8.8	11.2	7.2	62	Type1	1
1	31	5.2	6.8	10.9	4.2	33	Normal	0
2	32	6.8	8.8	11.2	7.2	62	Type1	1
3	21	5.7	5.8	10.7	4.8	49	Normal	0
4	33	6.8	8.8	11.2	7.2	62	Type1	1
...	...	...	...	...	...	...	...	...

```
pd.set_option('max_rows',16)
df
```



	Age	BS Fast	BS pp	Plasma R	Plasma F	HbA1c	Type	Class
0	50	6.8	8.8	11.2	7.2	62	Type1	1
1	31	5.2	6.8	10.9	4.2	33	Normal	0
2	32	6.8	8.8	11.2	7.2	62	Type1	1
3	21	5.7	5.8	10.7	4.8	49	Normal	0
4	33	6.8	8.8	11.2	7.2	62	Type1	1
...	...	...	...	...	...	...	...	...
1004	37	0.0	5.6	10.2	5.4	32	Normal	0
1005	23	6.3	4.2	12.2	7.8	57	Type2	2
1006	37	32.0	7.4	8.7	5.6	41	Normal	0
1007	46	19.0	6.3	7.9	3.9	40	Normal	0
1008	25	16.0	6.3	7.9	3.9	40	Normal	0

1009 rows × 8 columns

### Third Step:

We will drop the column of Type because we already have a Class Column showing the same information as Type column. So we will save the Type column in another list for future reference before dropping it

```
Typelist=df['Type']
df=df.drop('Type',axis=1)
```

Typelist

```
0    Type1
1    Normal
2    Type1
```

```
3      Normal
4      Type1
...
1004    Normal
1005    Type2
1006    Normal
1007    Normal
1008    Normal
Name: Type, Length: 1009, dtype: object
```

df

	Age	BS Fast	BS pp	Plasma R	Plasma F	HbA1c	Class
0	50	6.8	8.8	11.2	7.2	62	1
1	31	5.2	6.8	10.9	4.2	33	0
2	32	6.8	8.8	11.2	7.2	62	1
3	21	5.7	5.8	10.7	4.8	49	0
4	33	6.8	8.8	11.2	7.2	62	1
...	...	...	...	...	...	...	...
1004	37	0.0	5.6	10.2	5.4	32	0
1005	23	6.3	4.2	12.2	7.8	57	2
1006	37	32.0	7.4	8.7	5.6	41	0
1007	46	19.0	6.3	7.9	3.9	40	0
1008	25	16.0	6.3	7.9	3.9	40	0

1009 rows × 7 columns

Here the complete prerprocessing of the data ends.

Sorting Values by Age

```
df.sort_values(by='Age',inplace=True)
df
```

	Age	BS Fast	BS pp	Plasma R	Plasma F	HbA1c	Class
<b>525</b>	21	18.0	5.8	10.7	4.8	28	0
<b>190</b>	21	0.0	5.6	10.2	5.4	32	0
<b>196</b>	21	0.0	7.7	11.0	6.1	36	0
<b>465</b>	21	13.0	6.8	10.9	4.2	33	0
<b>200</b>	21	16.0	5.8	10.7	4.8	28	0



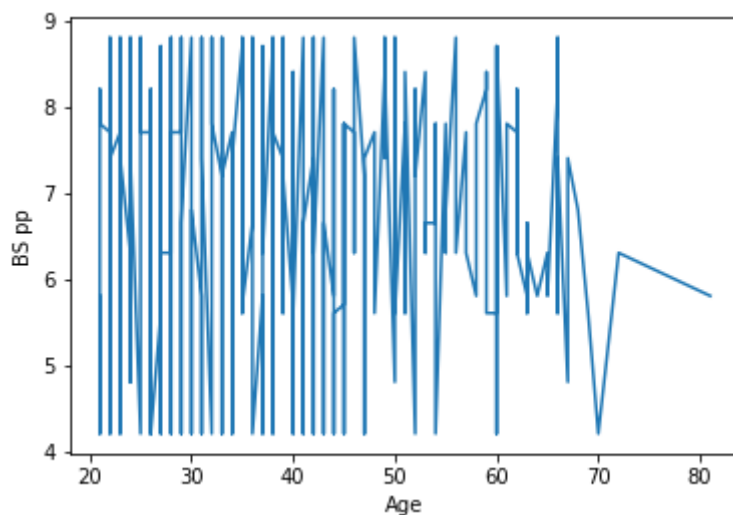
## Graph Plotting Stage

We will plot various graphs between different features of the dataset to see the figure of dataset and do a human-based analysis on which kind of models will be used to perfction on the dataset.

**459**    81    33.0    5.8    10.7    4.8    28    0

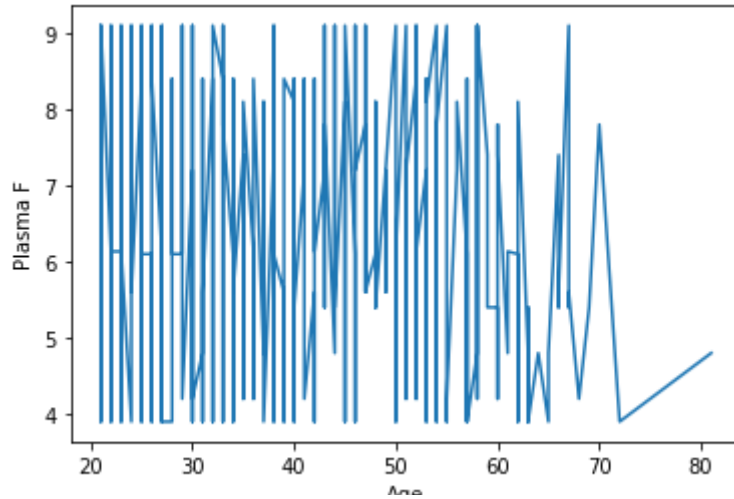
### Line Graph :

```
#age and bspp
agearr=df['Age']
BSpparr=df['BS pp']
plt.plot(agearr,BSpparr)
plt.xlabel('Age')
plt.ylabel('BS pp')
plt.show()
```



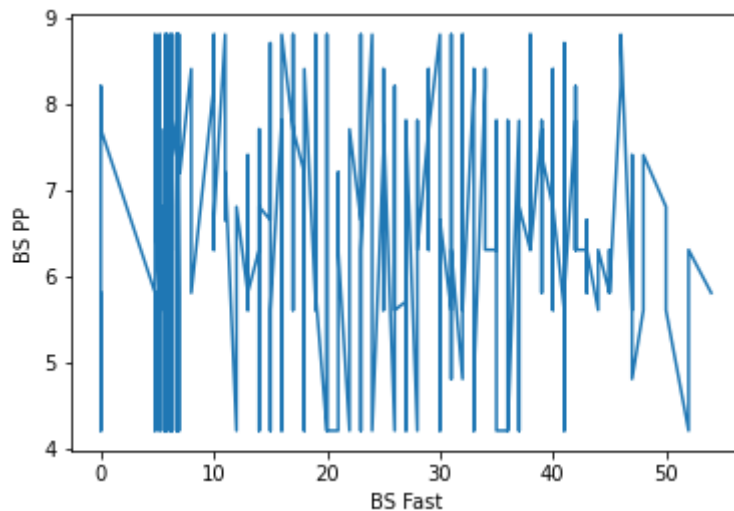
```
#age and plasma f
pfarr=df['Plasma F']
plt.plot(agearr,pfarr)
plt.xlabel('Age')
plt.ylabel('Plasma F')
plt.show
```

```
<function matplotlib.pyplot.show>
```



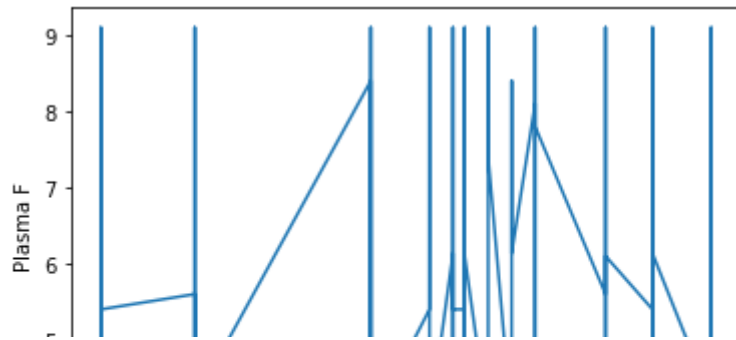
The two line graphs show we dont have direct dependence on Age Factor

```
#bs pp and bs fast
bsfastarr=df['BS Fast']
bsfastarr=bsfastarr.sort_values()
plt.plot(bsfastarr,BSpparr)
plt.xlabel('BS Fast')
plt.ylabel('BS PP')
plt.show()
```

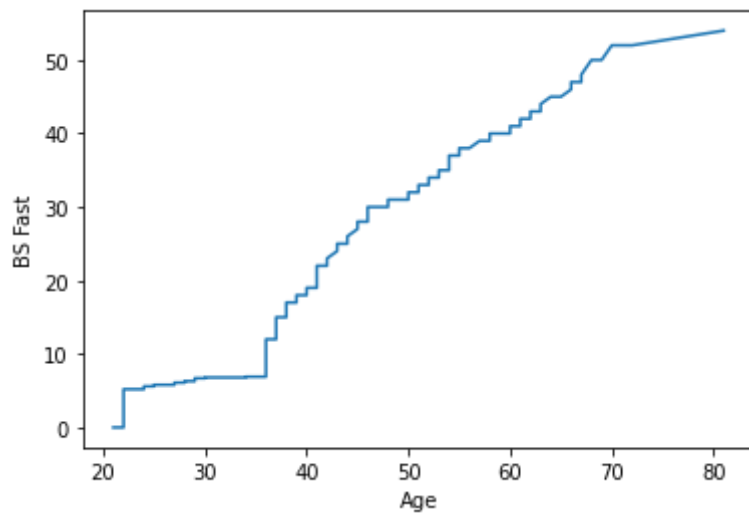


```
#plasma r and plasma f
prarr=df['Plasma R']
prarr=prarr.sort_values()
plt.plot(prarr,pfarr)
plt.xlabel('Plasma R')
plt.ylabel('Plasma F')
plt.show()
```



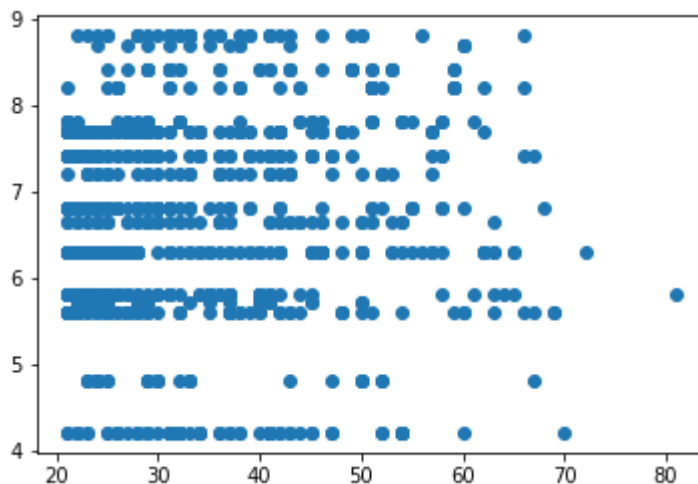


```
#age and bsfas
plt.plot(agearr,bsfastarr)
plt.xlabel('Age')
plt.ylabel('BS Fast')
plt.show()
```

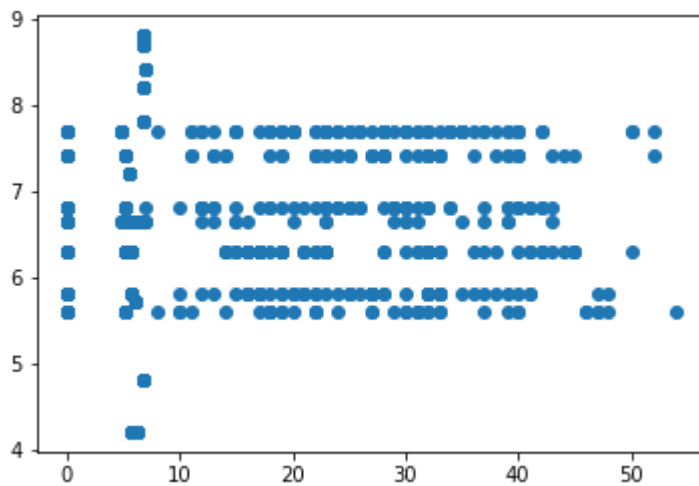


### Dot Graphs:

```
age1=df['Age']
bspp1=df['BS pp']
plt.scatter(age1,bspp1)
plt.show()
```

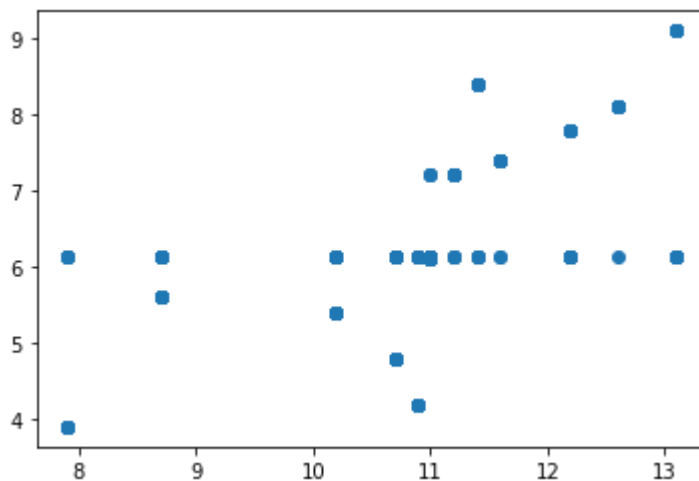


```
bsf1=df['BS Fast']
plt.scatter(bsf1,bspp1)
plt.show()
```



```
prarr=df['Plasma R']
pfarr=df['Plasma F']
plt.scatter(prarr,pfarr)
```

<matplotlib.collections.PathCollection at 0x7f09f75d0210>



### ***Test and Split Step :***

```
Xtrain,Xtest,Ytrain,Ytest=train_test_split(df[['Age','BS Fast','BS pp','Plasma R','Plasma
```

```
len(Xtrain), len(Ytrain)
```

```
(807, 807)
```

```
len(Xtest),len(Ytest)
```

```
(202, 202)
```

### ***Now The different models for Classification Start:***

## 1) Multinomial Logistic Regression:

```
regmnlr=LogisticRegression(multi_class='multinomial',solver='lbfgs')
regmnlr.fit(Xtrain,Ytrain)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Converge
```

```
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max\_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

[https://scikit-learn.org/stable/modules/linear\\_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

```
LogisticRegression(multi_class='multinomial')
```



```
regmnlr.predict(Xtest)
```

```
array([[0, 1, 0, 2, 0, 0, 0, 0, 0, 2, 0, 2, 0, 1, 0, 0, 2, 2, 0, 1, 0, 1,
        2, 0, 0, 0, 0, 0, 0, 1, 0, 1, 2, 2, 0, 0, 0, 0, 0, 1, 2, 0, 1, 2,
        1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 2, 0, 0, 0, 0, 0, 0, 2, 1, 1, 0,
        0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 2, 0, 0, 2, 0, 2, 0, 0,
        0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 2, 0, 2, 0,
        0, 0, 0, 0, 0, 0, 1, 2, 0, 0, 0, 2, 0, 0, 2, 0, 0, 0, 0, 2, 0, 0,
        2, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0, 0,
        2, 0, 0, 0, 0, 0, 0, 1, 0, 2, 2, 0, 2, 1, 0, 0, 1, 1, 1, 0, 0, 0,
        0, 0, 2, 2, 2, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0,
        0, 0, 2, 0]])
```

```
regmnlr.score(Xtest,Ytest)
```

```
0.9504950495049505
```

Multinomial Logistic Regression is one of the easiest Classification methods where we use modifications in the Logit function to classify for multi classes rather than binary classes. Here we acquired 94% Accuracy with 80% training data.

The lbfgs Solver is a limited memory solver. As a result it doesn't support one vs one kind of multinomial logistic regression. We have done one vs all multinomial logistic regression.

How does it work: The model first does a normal binary logistic classification on the data dividing it into two parts such that values with Class=0 go on one side and values with Class=1,2 go on the other side. After that we will do binary Logistic Regression on the mixed set and values having Class=1 will be classified differently from those having Class=2

## 2) Decision Tree Classifier (Gini Index) :

```
dtcgini= DecisionTreeClassifier()
dtcgini.fit(Xtrain,Ytrain)
```

```
DecisionTreeClassifier()
```

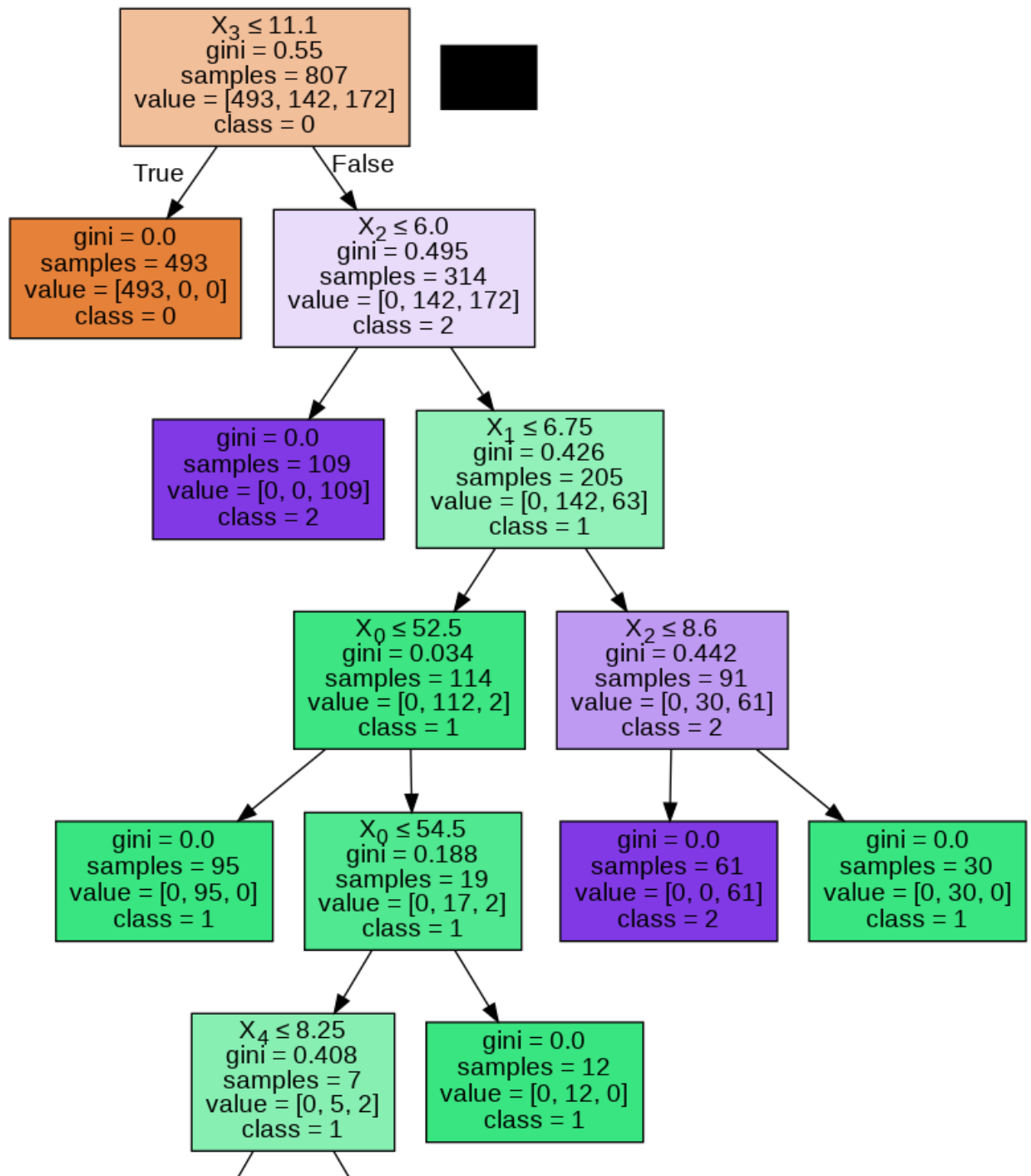
```
dtcgini.predict(Xtest)
```

```
array([[0, 1, 0, 2, 0, 0, 0, 0, 0, 2, 0, 2, 0, 1, 0, 0, 2, 2, 0, 1, 0, 1,
        2, 0, 0, 0, 0, 0, 0, 1, 0, 1, 2, 1, 0, 0, 0, 0, 0, 2, 1, 0, 1, 2,
        1, 0, 0, 0, 1, 0, 0, 1, 0, 2, 0, 2, 0, 0, 0, 0, 0, 0, 2, 2, 1, 0,
        0, 0, 0, 0, 0, 1, 0, 0, 2, 0, 0, 0, 0, 0, 2, 0, 0, 2, 0, 2, 0, 0,
        0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 2, 0, 2, 0,
        0, 0, 0, 0, 0, 0, 1, 2, 0, 0, 0, 1, 0, 0, 2, 0, 0, 0, 0, 2, 0, 0,
        1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0, 0,
        2, 0, 0, 0, 0, 0, 0, 1, 0, 2, 1, 0, 2, 1, 0, 0, 1, 1, 1, 0, 0, 0,
        0, 0, 1, 2, 2, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0,
        0, 0, 2, 0]])
```

```
dtcgini.score(Xtest,Ytest)
```

```
0.9801980198019802
```

```
dot_data = StringIO()
export_graphviz(dtcgini, out_file=dot_data,
                 filled=True, special_characters=True,class_names=['0','1','2'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes.png')
Image(graph.create_png())
```

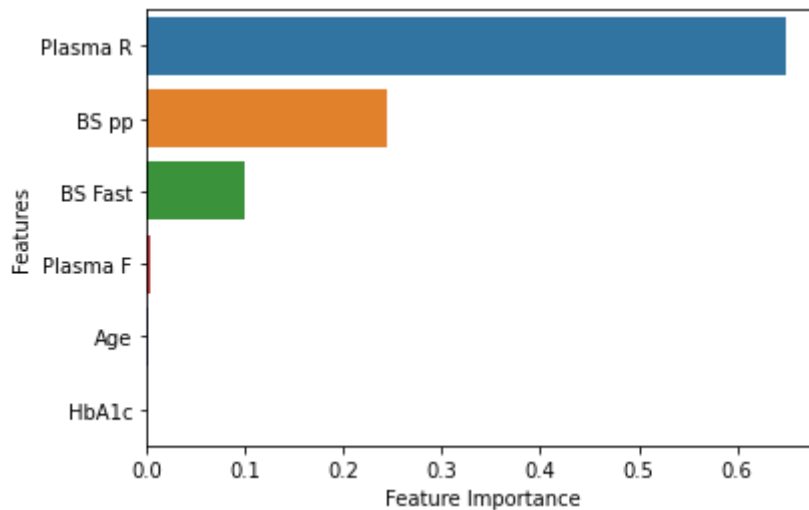


```
fimp=pd.Series(dtcgini.feature_importances_,index=Xtrain.columns).sort_values(ascending=False)
fimp
```

```
Plasma R    0.649765
BS pp      0.244291
BS Fast     0.100099
Plasma F    0.003431
Age         0.002415
HbA1c       0.000000
dtype: float64
```

```
class = 1 class = 2
```

```
srn.barplot(x=fimp,y=fimp.index)
plt.xlabel("Feature Importance")
plt.ylabel("Features")
plt.show()
```



Decision Tree Classifier acts as one of the more classic accurate algorithms for classifying multiclass data. Here we have used gini function (which uses probability square rather than log function) and we arrive at a 97.5% accuracy with 80% training data.

Gini Function :

$$\text{Gini}(D) = 1 - \sum_{i=1}^m P_i^2$$

Here :

D is a tuple

$P_i$  is the probability of D belonging to the class  $C_i$

m is the total number of classes

How it works: We use an Attribute Selection Method and choose the best attribute (here gini index). Make the attribute into decision node and break dataset into smaller subset as we go down the tree, the nodes go on purifying. The leaf nodes represent pure classes. The branches represent splitting decisions.

Here we can see how some of the features dont give significant input to the classification and hence as a result, Decision Tree Classifier using Gini Index cannot be called as the most reliable source of classification

### 3) Decision Tree Classifier (Information gain):

```
dtcig=DecisionTreeClassifier(criterion='entropy')
dtcig.fit(Xtrain,Ytrain)
```

```
DecisionTreeClassifier(criterion='entropy')
```

```
dtcig.predict(Xtest)
```

```
array([0, 1, 0, 2, 0, 0, 0, 0, 0, 2, 0, 2, 0, 1, 0, 0, 2, 2, 0, 1, 0, 1,
       2, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 2, 1, 0, 0, 0, 0, 0, 2, 1, 0, 1, 2,
```

```

1, 0, 0, 0, 1, 0, 0, 1, 0, 2, 0, 2, 0, 0, 0, 0, 0, 2, 2, 1, 0,
0, 0, 0, 0, 0, 1, 0, 0, 2, 0, 0, 0, 0, 0, 2, 0, 0, 2, 0, 2, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 2, 0, 2, 0,
0, 0, 0, 0, 0, 0, 1, 2, 0, 0, 0, 1, 0, 0, 2, 0, 0, 0, 2, 0, 0,
1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0, 0,
2, 0, 0, 0, 0, 0, 0, 1, 0, 2, 1, 0, 2, 1, 0, 0, 1, 1, 1, 0, 0, 0,
0, 0, 1, 2, 2, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 2, 2, 0, 0, 0, 0,
0, 0, 2, 0])

```

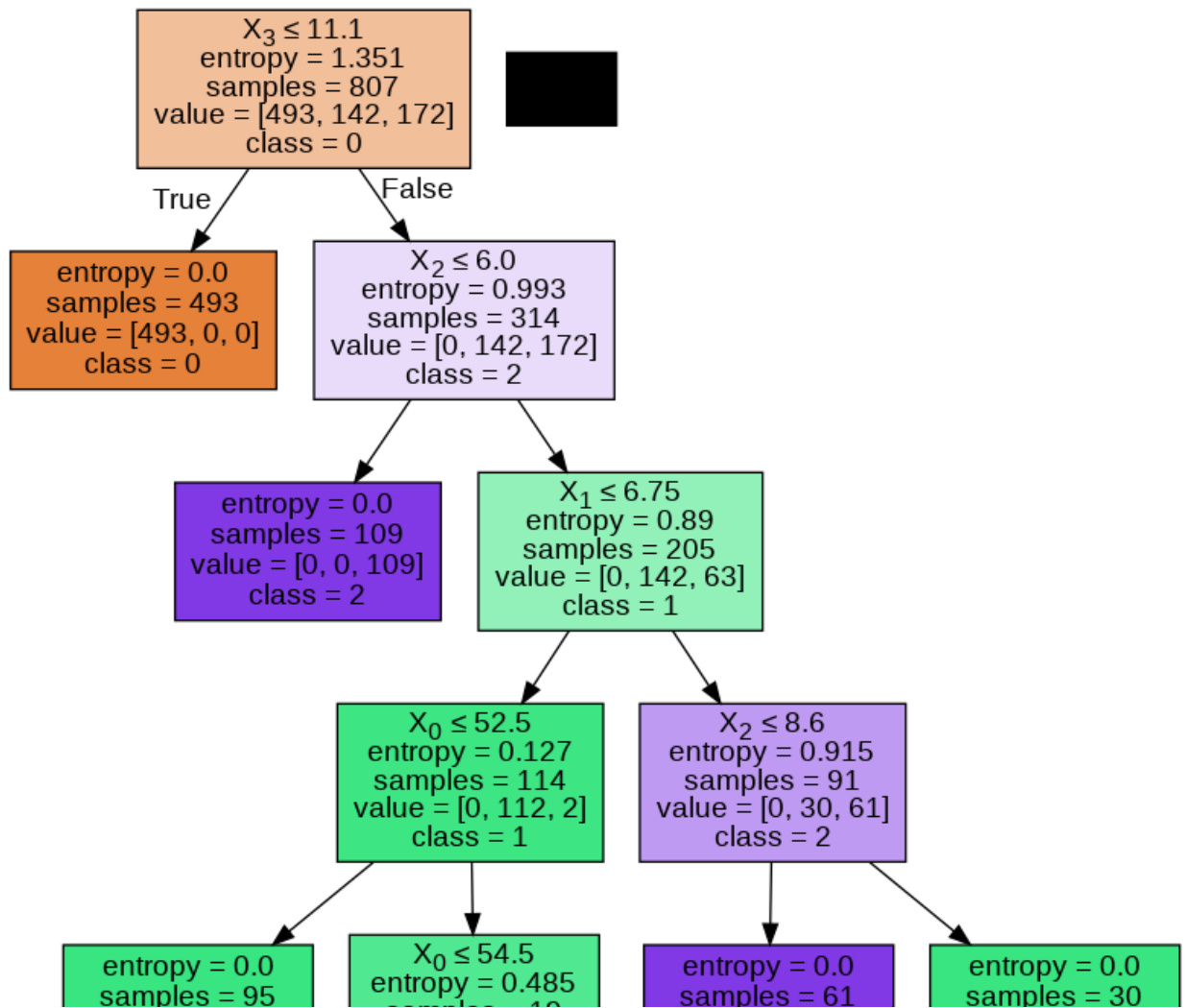
```
dtcig.score(Xtest,Ytest)
```

```
0.9801980198019802
```

```

dot_dataen = StringIO()
export_graphviz(dtcig, out_file=dot_dataen,
                filled=True, special_characters=True,class_names=['0','1','2'])
graphen = pydotplus.graph_from_dot_data(dot_dataen.getvalue())
graphen.write_png('diabetes.png')
Image(graphen.create_png())

```



```
fimp=pd.Series(dtcig.feature_importances_,index=Xtrain.columns).sort_values(ascending=False)
fimp
```

```
Plasma R    0.713839
BS pp       0.197653
BS Fast     0.077710
Age         0.007783
Plasma F    0.003015
HbA1c       0.000000
dtype: float64
```

```
srn.barplot(x=fimp,y=fimp.index)
plt.xlabel("Feature Importance")
plt.ylabel("Features")
plt.show()
```





Information Gain or Entropy is another way of using Decision Tree Classifier. Just as we used GINI index in the earlier model to choose the most accurate data splitting criterion, here Information Gain does the work and as we can see it has provided us with 100% result in 80% training data.

Formula for Entropy:

$$\text{Info}(D) = - \sum_{i=1}^m p_i \log_2 p_i$$

Here D is the tuple

Pi is the probability of D belonging to class Ci

m is the total number of classes.

Features such as Age, Plasma F and HbA1c give no input to prediction of classes of each data value. This makes Decision Tree using Information Gain highly unreliable and an algorithm that does not use all of the features given to it. Hence we cannot trust completely on the results predicted by Decision Tree using Information Gain

#### 4) Random Forest Classification :

```
clfrf=RandomForestClassifier(n_estimators=200)
clfrf.fit(Xtrain,Ytrain)
```

```
RandomForestClassifier(n_estimators=200)
```

```
clfrf.predict(Xtest)
```

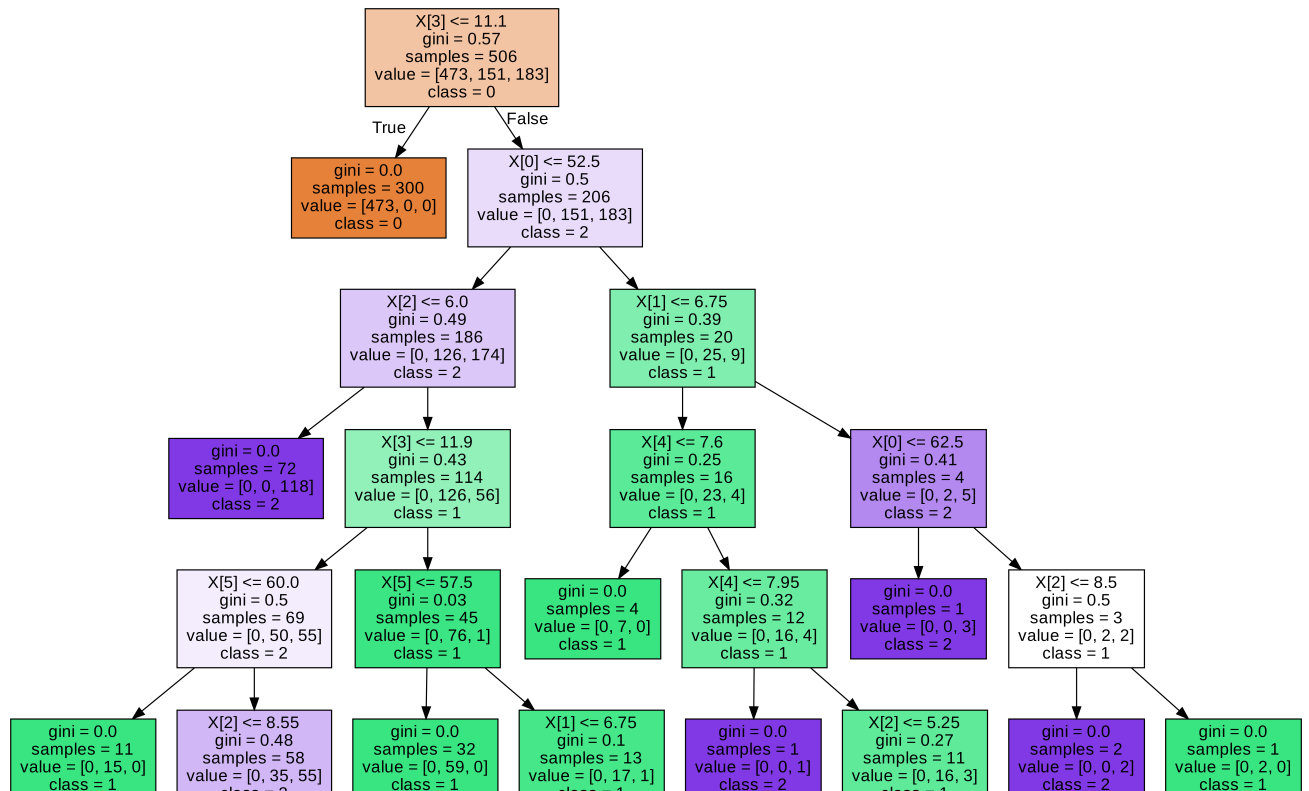
```
array([0, 1, 0, 2, 0, 0, 0, 0, 0, 2, 0, 2, 0, 1, 0, 0, 2, 2, 0, 1, 0, 1,
       2, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 2, 1, 0, 0, 0, 0, 0, 2, 1, 0, 1, 2,
       1, 0, 0, 0, 1, 0, 0, 1, 0, 2, 0, 2, 0, 0, 0, 0, 0, 0, 2, 2, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 2, 0, 0, 0, 0, 0, 2, 0, 0, 2, 0, 2, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 2, 0, 2, 0,
       0, 0, 0, 0, 0, 0, 1, 2, 0, 0, 0, 2, 0, 0, 2, 0, 0, 0, 0, 2, 0, 0,
       1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0, 0,
       2, 0, 0, 0, 0, 0, 0, 1, 0, 2, 1, 0, 2, 1, 0, 0, 1, 1, 1, 0, 0, 0,
       0, 0, 1, 2, 2, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0,
       0, 0, 2, 0])
```

```
clfrf.score(Xtest,Ytest)
```

```
0.9851485148514851
```

```
estimator = clfrf.estimators_[5]
export_graphviz(estimator, out_file='tree.dot')
```

```
export_graphviz(estimator, out_file='tree.dot',  
                class_names = ['0','1','2'], proportion = False,  
                precision = 2, filled = True)  
from subprocess import call  
call(['dot', '-Tpng', 'tree.dot', '-o', 'tree.png', '-Gdpi=600'])  
Image(filename = 'tree.png')
```



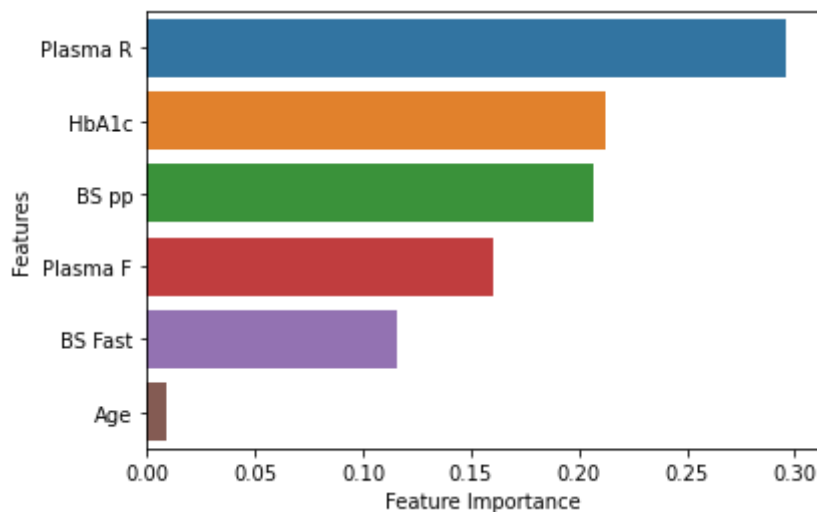
```
fimp=pd.Series(clfrf.feature_importances_,index=Xtrain.columns).sort_values(ascending=False)
fimp
```

```
Plasma R    0.295894
HbA1c       0.212245
BS pp       0.206924
Plasma F    0.160519
BS Fast     0.115728
Age         0.008690
dtype: float64
```

class = 1

class = 2

```
srn.barplot(x=fimp,y=fimp.index)
plt.xlabel("Feature Importance")
plt.ylabel("Features")
plt.show()
```



Random Forest is an Aggregation of many Decision Trees fromed from the subsets of the dataset provided to the algorithm as input. Decision Trees become highly ineefficient due to not using the data provided perfectly while depending on only certain features for prediction. Random Forest divides the dataset into different sub-datasets (also called Bootstrapped Datasets, in our case 200 bootstrapped datasets) and randomly selects a subset from feature set. Each bootstrapped dataset and feaure subset have equal number of instances and features respectively. We can see Random Forest as usually more accurate algorithm to Decision Tree as it is less sensitive to training data. We get almost 100% or 99.5% accuracy with 80% training dataset

How it Works:

Our dataset is divided into sub-datasets each having equal number of instances and each of them is worked upon with some feature subset. Each feature subset also has equal number of features which is usually equal to square root of total number of features. Each bootstrapped dataset is made into decision tree and all the final decision trees are aggregated to form one huge forest (or one huge decision tree)

### 5) Naive Bayes Algorithm :

```
modelnb=GaussianNB()
modelnb.fit(Xtrain,Ytrain)
```

```
GaussianNB()
```

```
pred=modelnb.predict(Xtest)
pred
```

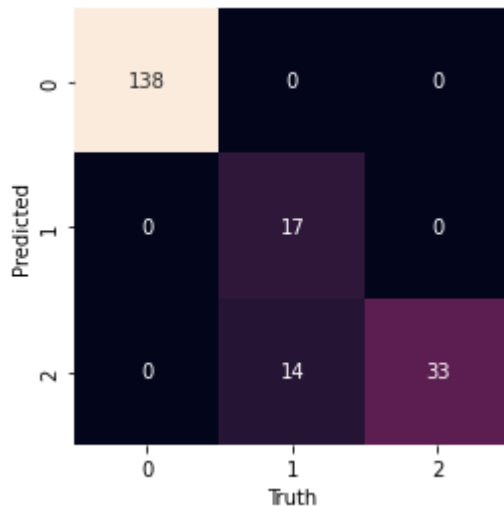
```
array([[0, 1, 0, 2, 0, 0, 0, 0, 0, 2, 0, 2, 0, 2, 0, 0, 2, 2, 0, 1, 0, 1,
        2, 0, 0, 0, 0, 0, 0, 2, 0, 1, 2, 2, 0, 0, 0, 0, 0, 2, 2, 0, 1, 2,
        2, 0, 0, 0, 1, 0, 0, 1, 0, 2, 0, 2, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0,
        0, 0, 0, 0, 0, 1, 0, 0, 2, 0, 0, 0, 0, 0, 0, 2, 0, 0, 2, 0, 2, 0, 0,
        0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 2, 0, 2, 0,
        0, 0, 0, 0, 0, 0, 1, 2, 0, 0, 0, 2, 0, 0, 2, 0, 0, 0, 0, 2, 0, 0,
        2, 0, 0, 2, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0, 0,
        2, 0, 0, 0, 0, 0, 0, 1, 0, 2, 2, 0, 2, 1, 0, 0, 2, 1, 2, 0, 0, 0,
        0, 0, 2, 2, 2, 0, 1, 0, 0, 0, 0, 2, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0,
        0, 0, 2, 0]])
```

```
modelnb.score(Xtest,Ytest)
```

```
0.9306930693069307
```

```
mat = confusion_matrix(pred, Ytest)
names = np.unique(pred)
srn.heatmap(mat, square=True, annot=True, fmt='d', cbar=False,
             xticklabels=names, yticklabels=names)
plt.xlabel('Truth')
plt.ylabel('Predicted')
```

Text(91.68, 0.5, 'Predicted')



The Naive Bayes algorithm works highly on conditional probability of the given feature in dataset. The Naive Bayes Algorithm:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

It fails on our classification dataset because our given values aren't continuous but each of the factor provides individual importance to prediction of Diabetes type. Hence calculation of individual probability to calculate the class of a given instance will be highly inefficient by the NB Standards. It gives us 94% accuracy with 80% training set.

How it works:

The algorithm will first calculate prior probability for each class (i.e. probability of each class). Then it will calculate the conditional probability of each feature with an assumption of its presence in each class. In case of Gaussian Naive Bayes algorithm, this is done by calculating mean and standard deviation and by plotting the Gaussian curve. For every new instance of data value to be predicted into class, each of the probabilities are predicted using The Gaussian curve and the probabilities are then multiplied. To discard the presence of very small probabilities we use logarithms on each of the probability. The higher aggregate conditional probability for a class will determine the data instance to be present in a class or not.

For eg. if the Aggregate Conditional Probability for a value for Class 0 is -114.56, for class 1 is -110.65 and for class 2 is -98.67, then the data instance belongs to class 2.

## 6) K Nearest Neighbours Classification with N neighbours as 5 :

```
knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(Xtrain,Ytrain)
```

```
KNeighborsClassifier()
```

```
predicted=knn.predict(Xtest)
predicted
```

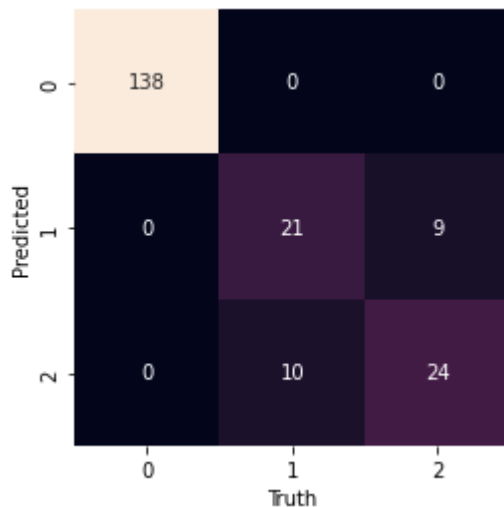
```
array([0, 1, 0, 2, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 2, 2, 0, 1, 0, 1,
       2, 0, 0, 0, 0, 0, 0, 1, 0, 1, 2, 2, 0, 0, 0, 0, 0, 2, 2, 0, 1, 2,
       2, 0, 0, 0, 1, 0, 0, 1, 0, 2, 0, 2, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 2, 0, 0, 0, 0, 0, 2, 0, 0, 1, 0, 2, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 2, 0, 2, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 2, 0, 0, 0, 0, 2, 0, 0,
       2, 0, 0, 2, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 2, 0, 2, 1, 0, 0, 1, 1, 2, 0, 0, 0,
       0, 0, 1, 2, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0,
       0, 0, 2, 0])
```

```
knn.score(Xtest,Ytest)
```

```
0.905940594059406
```

```
mat = confusion_matrix(predicted, Ytest)
names = np.unique(pred)
srn.heatmap(mat, square=True, annot=True, fmt='d', cbar=False,
             xticklabels=names, yticklabels=names)
plt.xlabel('Truth')
plt.ylabel('Predicted')
```

```
Text(91.68, 0.5, 'Predicted')
```



## 7) Support Vector Machine:

```
mac=svm.SVC(kernel='linear')
mac.fit(Xtrain,Ytrain)
```

```
SVC(kernel='linear')
```

```
macpr=mac.predict(Xtest)
macpr
```

```
array([0, 1, 0, 2, 0, 0, 0, 0, 0, 2, 0, 2, 0, 1, 0, 0, 2, 2, 0, 1, 0, 1,
       2, 0, 0, 0, 0, 0, 0, 1, 0, 1, 2, 1, 0, 0, 0, 0, 0, 2, 1, 0, 1, 2,
       1, 0, 0, 0, 1, 0, 0, 1, 0, 2, 0, 2, 0, 0, 0, 0, 0, 0, 2, 2, 1, 0,
```

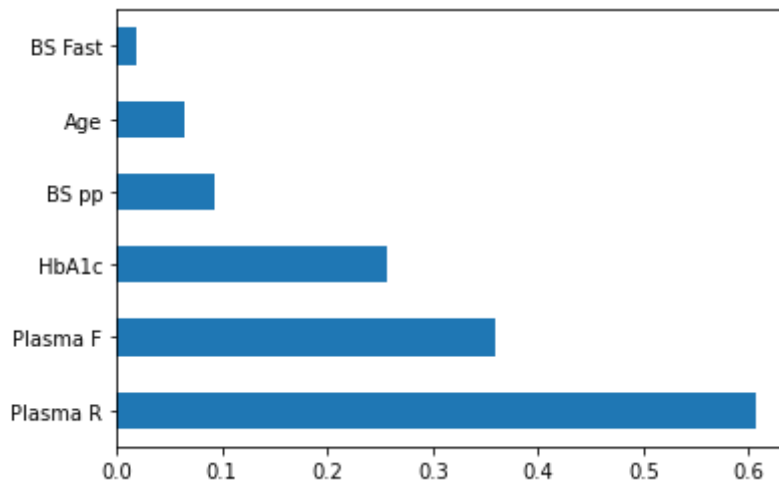
```
0, 0, 0, 0, 0, 1, 0, 0, 2, 0, 0, 0, 0, 0, 2, 0, 0, 2, 0, 2, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 2, 0, 2, 0,
0, 0, 0, 0, 0, 0, 1, 2, 0, 0, 0, 2, 0, 0, 2, 0, 0, 0, 0, 2, 0, 0,
1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0, 0,
2, 0, 0, 0, 0, 0, 0, 1, 0, 2, 1, 0, 2, 1, 0, 0, 1, 1, 1, 0, 0, 0,
0, 0, 2, 2, 2, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0,
0, 0, 2, 0])
```

```
mac.score(Xtest,Ytest)
```

```
0.9900990099009901
```

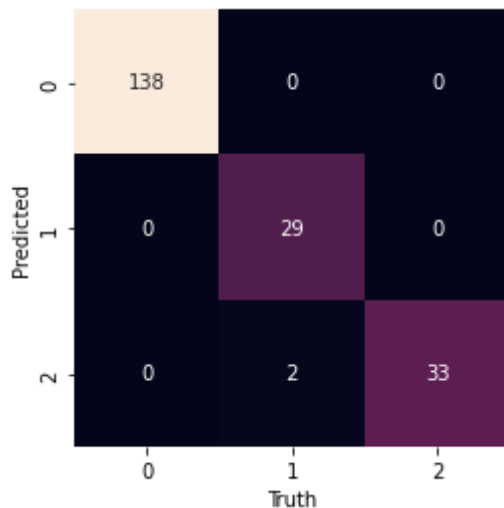
```
pd.Series(abs(mac.coef_[0]), index=Xtrain.columns).nlargest(10).plot(kind='barh')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f09f689ed10>
```



```
mat = confusion_matrix(macpr, Ytest)
names = np.unique(macpr)
srn.heatmap(mat, square=True, annot=True, fmt='d', cbar=False,
             xticklabels=names, yticklabels=names)
plt.xlabel('Truth')
plt.ylabel('Predicted')
```

```
Text(91.68, 0.5, 'Predicted')
```



```
df
```

	Age	BS Fast	BS pp	Plasma R	Plasma F	HbA1c	Class	
<b>525</b>	21	18.0	5.8	10.7	4.8	28	0	
<b>190</b>	21	0.0	5.6	10.2	5.4	32	0	
<b>196</b>	21	0.0	7.7	11.0	6.1	36	0	
<b>465</b>	21	13.0	6.8	10.9	4.2	33	0	
<b>200</b>	21	16.0	5.8	10.7	4.8	28	0	
...	...	...	...	...	...	...	...	
<b>684</b>	69	0.0	5.6	10.2	5.4	32	0	
<b>123</b>	69	5.2	5.6	10.2	5.4	32	0	
<b>666</b>	70	6.3	4.2	12.2	7.8	57	2	
<b>453</b>	72	0.0	6.3	7.9	3.9	40	0	
<b>459</b>	81	33.0	5.8	10.7	4.8	28	0	

1009 rows × 7 columns

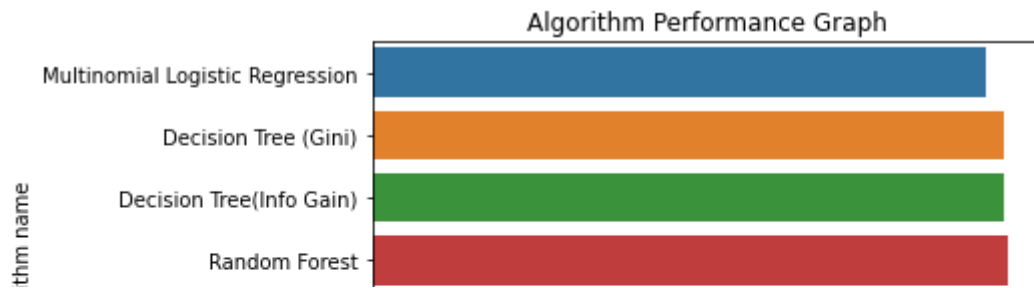
**Plotting a Bar Graph showing performance of each and every Algorithm so it is easy for analysis**

```
scores=np.array([regmnlr.score(Xtest,Ytest),dtcgini.score(Xtest,Ytest),dtcig.score(Xtest,Y
scores=100*scores
plotter= pd.Series(scores,index=['Multinomial Logistic Regression','Decision Tree (Gini)',
plotter
```

```
Multinomial Logistic Regression    95.049505
Decision Tree (Gini)                98.019802
Decision Tree(Info Gain)           98.019802
Random Forest                      98.514851
Naive Bayes                        93.069307
K Nearest Neighbors                90.594059
Support vector Machine              99.009901
dtype: float64
```

```
srn.barplot(x=plotter,y=plotter.index)
plt.title('Algorithm Performance Graph')
plt.xlabel('performance of Algorithm')
plt.ylabel('Algorithm name')
plt.show()
```





```
frzme=df[['Class']]
count=0
dfcop=df.drop(frzme,axis=1)
predvals=mac.predict(dfcop)
n=len(dfcop)
agearr=dfcop['Age']
for i in range(0,n):
    if(agearr[i]>20 and agearr[i]<=25):
        if(predvals[i]==1 or predvals[i]==2):
            count=count+1
perc=(count/n)*100
perc
```

12.487611496531219

✓ 0s completed at 11:28 AM

● ✕