```
import pandas as pd
import numpy as np
```

```
from google.colab import files
uploaded = files.upload()
```

Choose Files diabetes_da…upload.xlsx

- **diabetes_data_upload.xlsx**(application/vnd.openxmlformats-officedocument.spreadsheetml.sheet) - 43
modified: 10/18/2021 - 100% done

```
Saving diabetes_data_upload.xlsx to diabetes_data_upload (1).xlsx
```

```
import io
df = pd.read_excel(io.BytesIO(uploaded['diabetes_data_upload.xlsx']))
df
```

	Age	Gender	Polyuria	Polydipsia	sudden weight loss	weakness	Polyphagia	Genital thrush	visual blurring
0	40	Male	No	Yes	No	Yes	No	No	No
1	58	Male	No	No	No	Yes	No	No	Yes
2	41	Male	Yes	No	No	Yes	Yes	No	No
3	45	Male	No	No	Yes	Yes	Yes	Yes	No
4	60	Male	Yes	Yes	Yes	Yes	Yes	No	Yes
...	...	...	...	...	...	...	...	...	...
515	39	Female	Yes	Yes	Yes	No	Yes	No	No
516	48	Female	Yes	Yes	Yes	Yes	Yes	No	No
517	58	Female	Yes	Yes	Yes	Yes	Yes	No	Yes
518	32	Female	No	No	No	Yes	No	No	Yes
519	42	Male	No	No	No	No	No	No	No

520 rows × 17 columns

```
df.dtypes
```

```
Age                    int64
Gender                object
Polyuria              object
Polydipsia            object
sudden weight loss    object
weakness              object
Polyphagia            object
Genital thrush        object
visual blurring       object
```

```
Itching               object
Irritability          object
delayed healing       object
partial paresis       object
muscle stiffness      object
Alopecia              object
Obesity               object
class                 object
dtype: object
```

```
print(df.isnull().values.any())
```

```
False
```

There are no missing values.

```
df.info
```

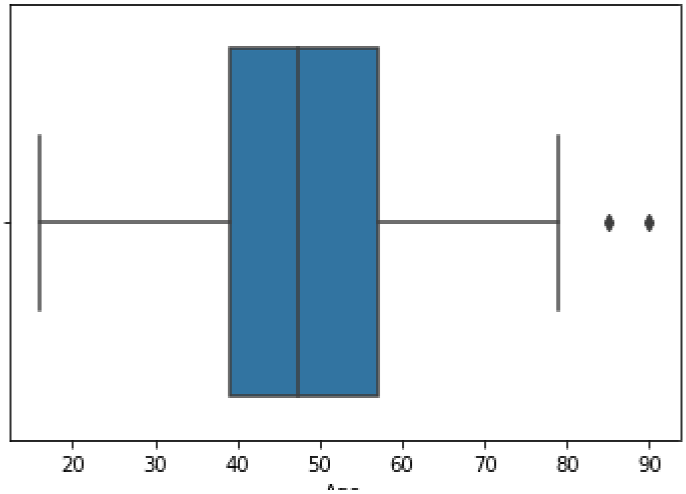
```
<bound method DataFrame.info of      Age  Gender Polyuria  ... Alopecia Obesity     clas
0     40    Male       No  ...      Yes     Yes  Positive
1     58    Male       No  ...      Yes      No  Positive
2     41    Male      Yes  ...      Yes      No  Positive
3     45    Male       No  ...       No      No  Positive
4     60    Male      Yes  ...      Yes     Yes  Positive
..   ...     ...      ...  ...      ...     ...       ...
515   39  Female      Yes  ...       No      No  Positive
516   48  Female      Yes  ...       No      No  Positive
517   58  Female      Yes  ...       No     Yes  Positive
518   32  Female       No  ...      Yes      No  Negative
519   42    Male       No  ...       No      No  Negative

[520 rows x 17 columns]>
```

```
df.describe(include='object')
```

	Gender	Polyuria	Polydipsia	sudden weight loss	weakness	Polyphagia	Genital thrush	visual blurring	I
count	520	520	520	520	520	520	520	520	
unique	2	2	2	2	2	2	2	2	
top	Male	No	No	No	Yes	No	No	No	
freq	328	262	287	303	305	283	404	287	

```
import seaborn as sns
sns.boxplot(x=df['Age'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f700982af50>
```



Any age over 80 are outliers. We can proceed to remove them.

```
df_new = df.drop(df[df.Age > 80].index)
```

```
df_new.describe
```

```
<bound method NDFrame.describe of      Age  Gender Polyuria  ... Alopecia Obesity    c
0     40    Male       No  ...      Yes     Yes  Positive
1     58    Male       No  ...      Yes      No  Positive
2     41    Male      Yes  ...      Yes      No  Positive
3     45    Male       No  ...       No      No  Positive
4     60    Male      Yes  ...      Yes     Yes  Positive
..   ...     ...      ...  ...      ...     ...       ...
515   39  Female      Yes  ...       No      No  Positive
516   48  Female      Yes  ...       No      No  Positive
517   58  Female      Yes  ...       No     Yes  Positive
518   32  Female       No  ...      Yes      No  Negative
519   42    Male       No  ...       No      No  Negative

[516 rows x 17 columns]>
```

Rows with the outliers are removed and dataset is renamed to be df_new

First, I will try to deploy classification algorithm but I will convert all the string value to numerical value first

```
d = {'Male': 1, 'Female': 0}
df_new['Gender'] = df_new['Gender'].map(d)
d = {'Yes': 1, 'No': 0}
df_new['Polyuria'] = df_new['Polyuria'].map(d)
df_new['Polydipsia'] = df_new['Polydipsia'].map(d)
df_new['sudden weight loss'] = df_new['sudden weight loss'].map(d)
```

```
df_new['weakness'] = df_new['weakness'].map(d)
df_new['Polyphagia'] = df_new['Polyphagia'].map(d)
df_new['Genital thrush'] = df_new['Genital thrush'].map(d)
df_new['visual blurring'] = df_new['visual blurring'].map(d)
df_new['Itching'] = df_new['Itching'].map(d)
df_new['Irritability'] = df_new['Irritability'].map(d)
df_new['delayed healing'] = df_new['delayed healing'].map(d)
df_new['partial paresis'] = df_new['partial paresis'].map(d)
df_new['muscle stiffness'] = df_new['muscle stiffness'].map(d)
df_new['Alopecia'] = df_new['Alopecia'].map(d)
df_new['Obesity'] = df_new['Obesity'].map(d)
d = {'Positive': 1, 'Negative': 0}
df_new['class'] = df_new['class'].map(d)
df_new
```

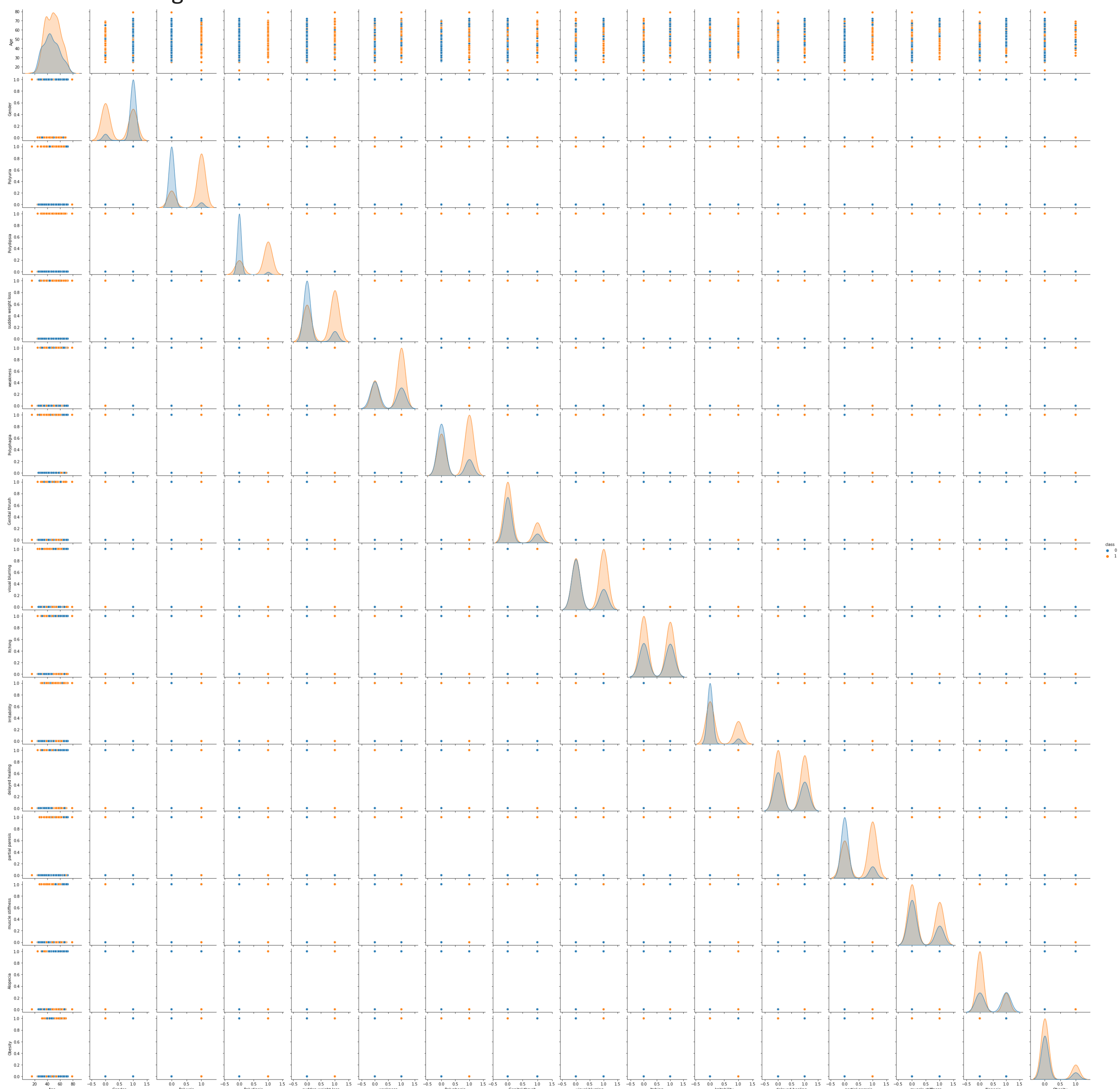
	Age	Gender	Polyuria	Polydipsia	sudden weight loss	weakness	Polyphagia	Genital thrush	visual blurring
0	40	1	0	1	0	1	0	0	0
1	58	1	0	0	0	1	0	0	1
2	41	1	1	0	0	1	1	0	0
3	45	1	0	0	1	1	1	1	0
4	60	1	1	1	1	1	1	0	1
...	...	...	...	...	...	...	...	...	...
515	39	0	1	1	1	0	1	0	0
516	48	0	1	1	1	1	1	0	0
517	58	0	1	1	1	1	1	0	1
518	32	0	0	0	0	1	0	0	1
519	42	1	0	0	0	0	0	0	0

516 rows × 17 columns

Let's do some more visualization on the data

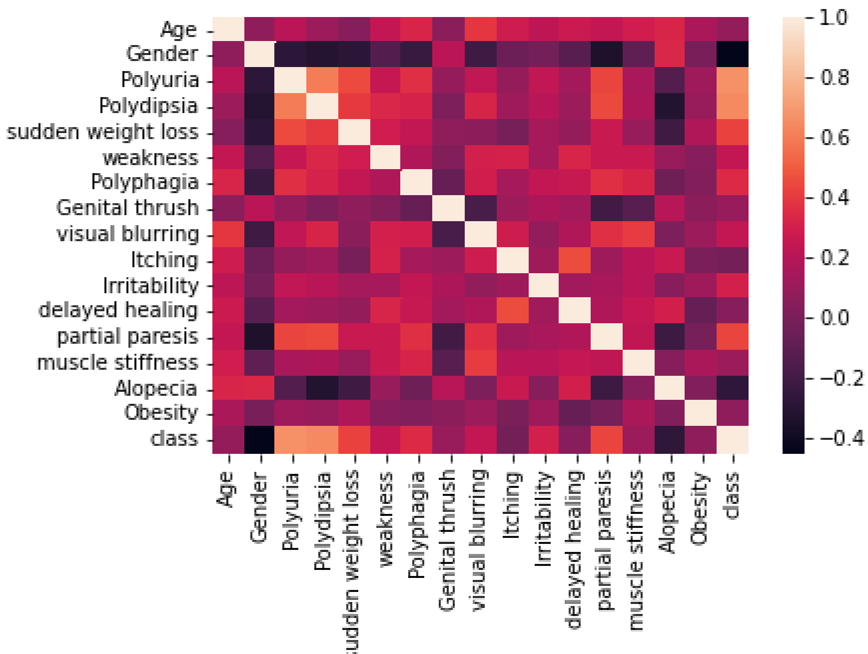
```
import matplotlib.pyplot as plt
sns.pairplot(data=df_new, hue = 'class')
```

```
<seaborn.axisgrid.PairGrid at 0x7f6fee846690>
```



```
sns.heatmap(df_new.corr())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6ff9529d90>
```



Going to prepare for the decision tree

```
X = df_new.drop(['class'], axis=1)
X
```

	Age	Gender	Polyuria	Polydipsia	sudden weight loss	weakness	Polyphagia	Genital thrush	visual blurring
0	40	1	0	1	0	1	0	0	0
1	58	1	0	0	0	1	0	0	1
2	41	1	1	0	0	1	1	0	0
3	45	1	0	0	1	1	1	1	0
4	60	1	1	1	1	1	1	0	1
...	...	...	...	...	...	...	...	...	...
515	39	0	1	1	1	0	1	0	0
516	48	0	1	1	1	1	1	0	0

```
Y = df_new['class']
Y
```

```
0      1
1      1
2      1
3      1
4      1
      ..
515    1
516    1
517    1
518    0
519    0
Name: class, Length: 516, dtype: int64
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, random_state=1)
```

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

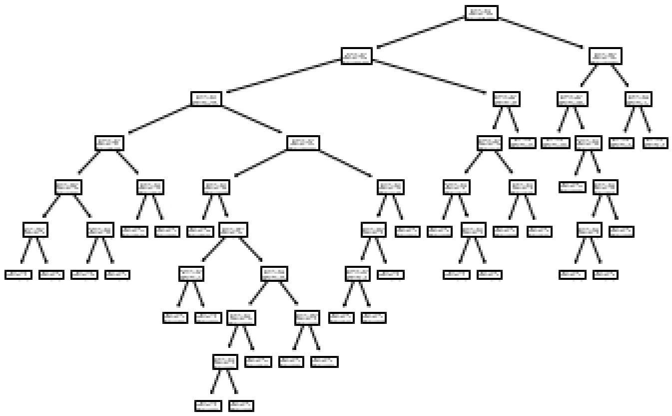
```
dtree = DecisionTreeClassifier()
dtree = dtree.fit(x_train, y_train)
```

```
from sklearn import tree
```

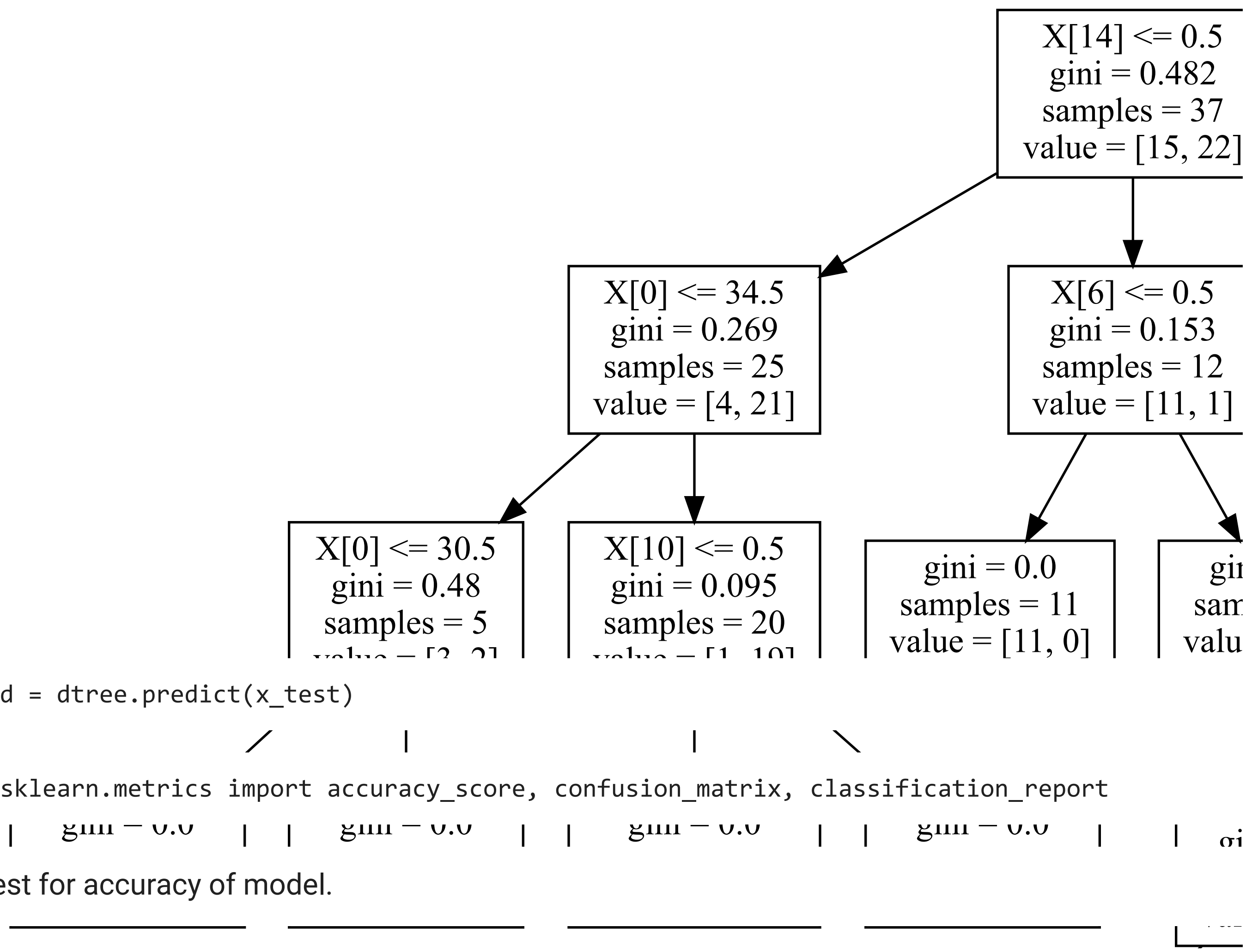
```
tree.plot_tree(dtree)
```

```
 Text(102.30000000000001, 163.07999999999998, 'X[1] <= 0.5\ngini = 0.347\nsamples = 148\
 Text(53.733333333333334, 141.336, 'X[14] <= 0.5\ngini = 0.482\nsamples = 37\nvalue = [15
 Text(33.06666666666667, 119.592, 'X[0] <= 34.5\ngini = 0.269\nsamples = 25\nvalue = [4,
 Text(16.533333333333335, 97.848, 'X[0] <= 30.5\ngini = 0.48\nsamples = 5\nvalue = [3, 2
 Text(8.266666666666667, 76.10399999999998, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(24.800000000000004, 76.10399999999998, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(49.60000000000001, 97.848, 'X[10] <= 0.5\ngini = 0.095\nsamples = 20\nvalue = [1,
 Text(41.333333333333336, 76.10399999999998, 'gini = 0.0\nsamples = 19\nvalue = [0, 19]
 Text(57.866666666666674, 76.10399999999998, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(74.4, 119.592, 'X[6] <= 0.5\ngini = 0.153\nsamples = 12\nvalue = [11, 1]'),
 Text(66.13333333333334, 97.848, 'gini = 0.0\nsamples = 11\nvalue = [11, 0]'),
 Text(82.66666666666667, 97.848, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(150.86666666666667, 141.336, 'X[10] <= 0.5\ngini = 0.179\nsamples = 111\nvalue =
 Text(107.46666666666668, 119.592, 'X[5] <= 0.5\ngini = 0.097\nsamples = 98\nvalue = [93
 Text(99.20000000000002, 97.848, 'gini = 0.0\nsamples = 56\nvalue = [56, 0]'),
 Text(115.73333333333335, 97.848, 'X[0] <= 38.0\ngini = 0.21\nsamples = 42\nvalue = [37,
 Text(95.06666666666668, 76.10399999999998, 'X[14] <= 0.5\ngini = 0.49\nsamples = 7\nval
 Text(86.80000000000001, 54.360000000000014, 'gini = 0.0\nsamples = 4\nvalue = [4, 0]'),
 Text(103.33333333333334, 54.360000000000014, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]')
 Text(136.4, 76.10399999999998, 'X[4] <= 0.5\ngini = 0.108\nsamples = 35\nvalue = [33, 2
 Text(119.86666666666667, 54.360000000000014, 'X[9] <= 0.5\ngini = 0.062\nsamples = 31\n
 Text(111.60000000000001, 32.615999999999985, 'X[8] <= 0.5\ngini = 0.375\nsamples = 4\nv
 Text(103.33333333333334, 10.872000000000014, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]')
 Text(119.86666666666667, 10.872000000000014, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]')
 Text(128.13333333333335, 32.615999999999985, 'gini = 0.0\nsamples = 27\nvalue = [27, 0]
 Text(152.93333333333334, 54.360000000000014, 'X[0] <= 49.5\ngini = 0.375\nsamples = 4\n
 Text(144.66666666666669, 32.615999999999985, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]')
 Text(161.20000000000002, 32.615999999999985, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]')
 Text(194.26666666666668, 119.592, 'X[7] <= 0.5\ngini = 0.497\nsamples = 13\nvalue = [7,
 Text(186.00000000000003, 97.848, 'X[0] <= 42.5\ngini = 0.346\nsamples = 9\nvalue = [7,
 Text(177.73333333333335, 76.10399999999998, 'X[13] <= 0.5\ngini = 0.444\nsamples = 3\nv
 Text(169.4666666666667, 54.360000000000014, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(186.00000000000003, 54.360000000000014, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]')
 Text(194.26666666666668, 76.10399999999998, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]'),
 Text(202.53333333333336, 97.848, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(252.13333333333335, 163.07999999999998, 'X[10] <= 0.5\ngini = 0.358\nsamples = 30\
 Text(243.8666666666667, 141.336, 'X[13] <= 0.5\ngini = 0.492\nsamples = 16\nvalue = [7,
 Text(227.33333333333334, 119.592, 'X[12] <= 0.5\ngini = 0.219\nsamples = 8\nvalue = [1,
 Text(219.0666666666667, 97.848, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]'),
 Text(235.60000000000002, 97.848, 'X[5] <= 0.5\ngini = 0.444\nsamples = 3\nvalue = [1, 2
 Text(227.33333333333334, 76.10399999999998, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(243.8666666666667, 76.10399999999998, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(260.40000000000003, 119.592, 'X[8] <= 0.5\ngini = 0.375\nsamples = 8\nvalue = [6,
 Text(252.13333333333335, 97.848, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(268.6666666666667, 97.848, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]'),
 Text(260.40000000000003, 141.336, 'gini = 0.0\nsamples = 14\nvalue = [0, 14]'),
 Text(301.73333333333335, 184.824, 'X[0] <= 69.5\ngini = 0.144\nsamples = 167\nvalue =
 Text(285.20000000000005, 163.07999999999998, 'X[15] <= 0.5\ngini = 0.095\nsamples = 160
 Text(276.93333333333334, 141.336, 'gini = 0.0\nsamples = 121\nvalue = [0, 121]'),
 Text(293.4666666666667, 141.336, 'X[11] <= 0.5\ngini = 0.326\nsamples = 39\nvalue = [8,
 Text(285.20000000000005, 119.592, 'gini = 0.0\nsamples = 24\nvalue = [0, 24]'),
 Text(301.73333333333335, 119.592, 'X[3] <= 0.5\ngini = 0.498\nsamples = 15\nvalue = [8,
 Text(293.4666666666667, 97.848, 'X[1] <= 0.5\ngini = 0.198\nsamples = 9\nvalue = [8, 1]
 Text(285.20000000000005, 76.10399999999998, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(301.73333333333335, 76.10399999999998, 'gini = 0.0\nsamples = 8\nvalue = [8, 0]'),
 Text(310.00000000000006, 97.848, 'gini = 0.0\nsamples = 6\nvalue = [0, 6]'),
 Text(318.2666666666667, 163.07999999999998, 'X[8] <= 0.5\ngini = 0.408\nsamples = 7\nva
```

```
 Text(310.00000000000006, 141.336, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(326.53333333333336, 141.336, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]')]
```



```
import graphviz
dot_data = tree.export_graphviz(dtree, out_file=None)
graph = graphviz.Source(dot_data)
graph
```




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

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

We test for accuracy of model.

```
print("Accuracy for Simple Classification Tree:",accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9883040935672515
```

We will try using Random Forest classifer to see compare the accuracy since it is an extension of decision tree with more complication

```
from sklearn.ensemble import RandomForestClassifier
```

```
model = RandomForestClassifier(random_state=1)
```

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

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=1, verbose=0,
                       warm_start=False)
```

```
y_pred2 = model.predict(x_test)
```

```
print("Accuracy for Random Forest:",accuracy_score(y_test, y_pred))
```

```
Accuracy for Random Forest: 0.9883040935672515
```

Since the two accuracy is the same. I believe the first decision tree was the final decision tree Random Forest Classifier chose as the best decision tree.

Now we will look at confusion matrix.

```
confusion_matrix(y_test, y_pred)
```

```
array([[ 64,   1],
       [  1, 105]])
```

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

```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98        65
           1       0.99      0.99      0.99       106

    accuracy                           0.99       171
   macro avg       0.99      0.99      0.99       171
```

```
weighted avg       0.99      0.99      0.99       171
```

Now I will try to use Logistic Regression

```
from sklearn.linear_model import LogisticRegression
```

```
logreg = LogisticRegression()
logreg.fit(x_train, y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
y_predreg = logreg.predict(x_test)
print("Accuracy for Logistic Regression:",accuracy_score(y_test, y_predreg))
```

```
Accuracy for Logistic Regression: 0.9707602339181286
```

```
confusion_matrix(y_test, y_predreg)
```

```
array([[ 63,   2],
       [  3, 103]])
```

```
print(classification_report(y_test, y_predreg))
```

```
              precision    recall  f1-score   support

           0       0.95      0.97      0.96        65
           1       0.98      0.97      0.98       106

    accuracy                           0.97       171
   macro avg       0.97      0.97      0.97       171
weighted avg       0.97      0.97      0.97       171
```

We can now compare decision tree vs logistic regression and we can see that using decision tree is a better option