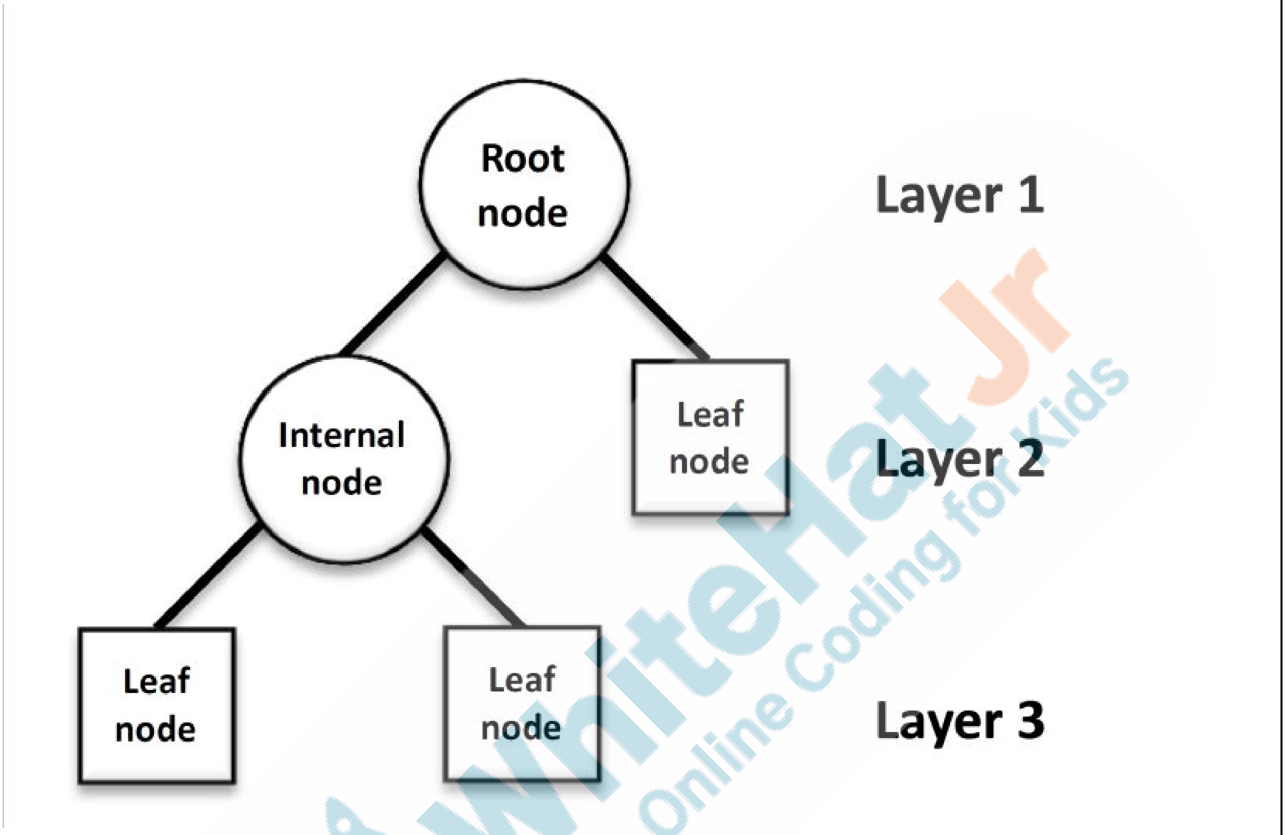
Topic	**Decision Tree**	
Class Description	**Students learns to create a Decision Tree algorithm and plot the Decision Tree chart**	
Class	**C119**	
Class time	**45 mins**	
Goal	● Learn about Decision Tree Algorithm. ● Write a Decision Tree Algorithm. ● Create a Decision Tree chart	
Resources Required	● Teacher Resources ○ Google Colab Note book ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ● Student Resources ○ Google Colab Notebook ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen	
Class structure	**Warm Up** **Teacher-led Activity** **Student-led Activity** **Wrap up**	**5 mins** **15 min** **15 min** **5 min**

<u>CONTEXT</u>

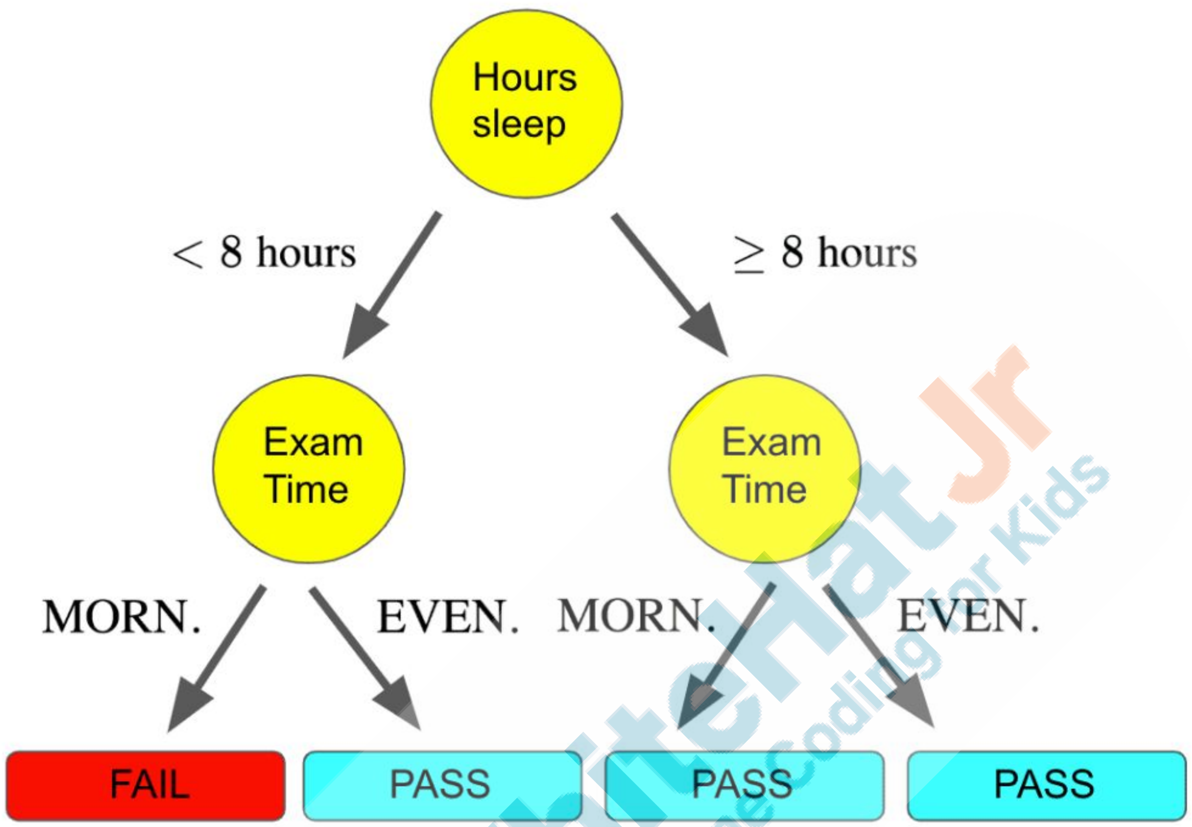
- **Introduce the concept of Decision Tree.**

Class Steps	**Teacher Action**	**Student Action**
Step 1: **Warm Up** **(5 mins)**	Hi <Student Name> Let's revise what we did in last class	ESR:- -We studied about clustering.

		-We saw how a data is grouped and analysed.
	Till now we seen the unsupervised machine learning algorithms. Today we'll learn about a supervised machine learning algorithm and that is the Decision Tree. What can you understand from the name Decision Tree?	ESR: Varied!
	Decision tree means taking the further decisions based on the results got from the previous prediction. Let's learn more about this in detail.	-
Teacher Initiates Screen Share		
CHALLENGE ● **Explore Decision Tree algorithm** ● **Create a chart based on Decision Tree algorithm**		
Step 2: Teacher-led Activity (15 min)	One of the most commonly used Machine Learning Algorithm is the Decision Tree, which is a flow chart like structure that leads us to an outcome based on the data and the decisions it takes. A typical decision tree diagram (flow chart) looks like this : <Teacher opens the link and shows the image> https://www.researchgate.net/profile/Mei-Hung_Chiu/publication/295860754/figure/fig3/AS:333010919542789@1456407398669/Basic-structure-of-a-	-

	decision-tree-All-decision-trees-are-built-through-recursion.png	
 Root node Layer 1 Internal node Leaf node Layer 2 Leaf node Leaf node Layer 3		
	Have you seen this structure before? Yes. This structure is called as Decision tree.Decision trees provide an effective method of Decision Making because they: Clearly lay out the problem so that all options can be challenged. Allowing us to analyze fully the possible consequences of a decision. Provide a framework to quantify the values of outcomes and the probabilities of achieving them.	ESR: Yes , It looks like a family tree.

	Can you read the the components for me? Very Good. -**Root Node/ Decision Node** - The root node is also called as decision node and is the one which represents the entire population. This is the point from where the population gets divided into 2 or more groups. **-Internal Node** - An internal node is again like the root node, but it does not contain the entire population. We further divide our data into more groups from here. **-Leaf Node** - A leaf node is the one that represents the final outcome.	ESR: Yes, They are Root Node, Internal Node, Leaf Node
	Let's understand this with an example. <Teacher opens the link and shows the image> https://www.mihaileric.com/static/layer2TreeDiagram-95dec8fbb247ce5161f63e63d8816fed-28303.png What can you make our from this image?	ESR: We can see a decision tree where the decision is made on the basis of the number of hours the student sleeps.

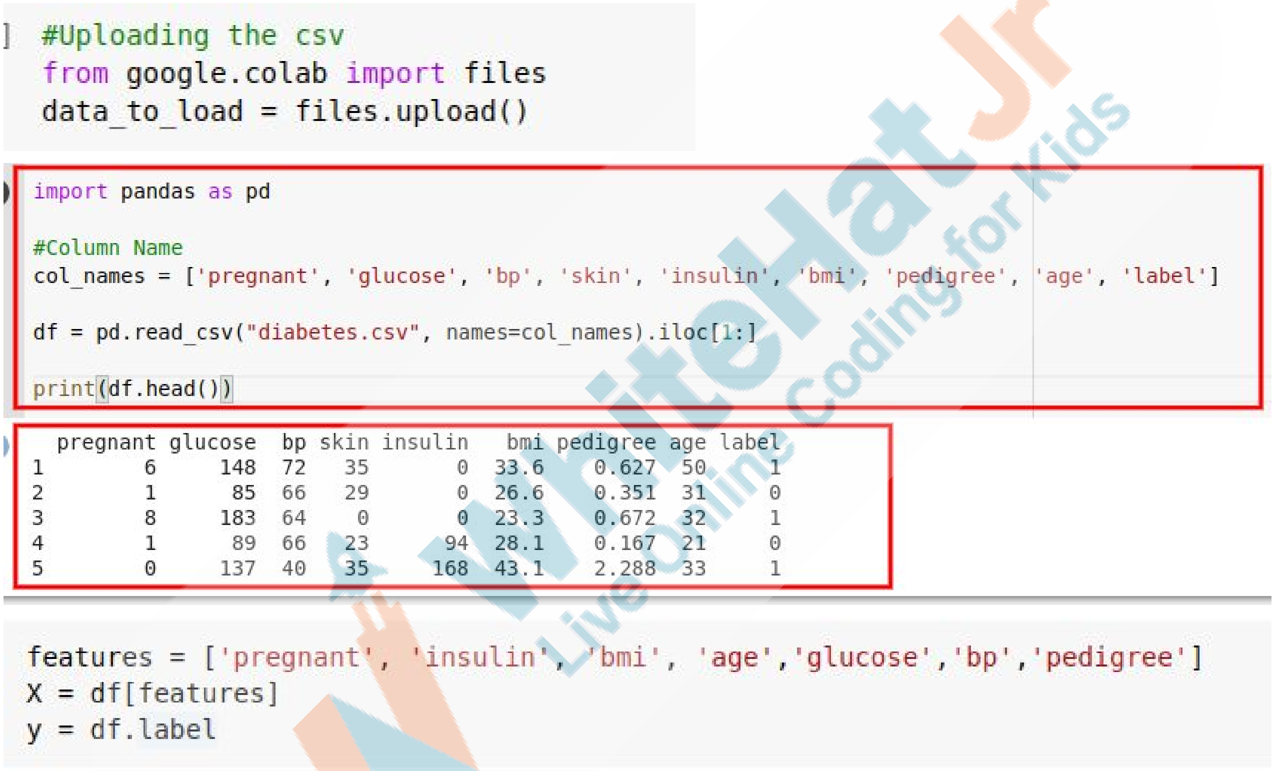


	Perfect!. Technically speaking we can say that the root node has all the population. It decides to split the data based on the number of hours of sleep. After splitting the data, It has the internal nodes as the population of the students who slept for less than 8 hours on the left and the students who slept for more than or equal to 8 hours on the right. Now it further splits the population	Student observes and learns

	more based on the time of their exam, if it is in the morning or in the evening. Based on the analysis from this decision tree, we can say that a student who sleeps for less than 8 hours and has their exam in the morning would fail.	
	Now let's see how the decision tree algorithm works. The first thing that would come in mind is that, how do we split the data? What is the best metric to split the data? In the example above, what could have been the measure of splitting the data? Yes! For this we have something known as Attribute Selection Measures or ASM which we use to split the data.	ESR: In the above example the data can be split based on the time of the exam
	Attribute Selection Measures or **ASM** It is used for selecting the splitting criteria that splits data in the best possible manner. It provides a rank to each feature by explaining the given dataset. The feature with the best score gets selected as the splitting attribute. Next, based on the feature that is selected, our algorithm would split the data into 2 or more groups.	-

	It starts building a tree structure by repeating this process recursively for each child (or Internal Node) until it reached a final output following all the paths in the flow chart.	
	Let's look at some code. <Teacher opens the colab notebook from the Teacher Activity1> <Teacher downloads the code from the Teacher Activity 2>	-
	We are using the data of diabetes patients depending on multiple varibles. <Teacher uploads the data in Colab Notebook> <Teacher codes to create a data frame> Code:- **import pandas as pd** **#Column Name** **col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']** **df = pd.read_csv("diabetes.csv", names=col_names).iloc[1:]** **print(df.head())** We'll also create 2 diiferent dataframes . 1 with all the variables and 2nd with label. <Teacher codes to create 2 different	Student helps teacher with the code

	dataframes to features and label variable.> Code:- **features = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']** **X = df[features]** **y = df.label**	



```
#Uploading the csv
from google.colab import files
data_to_load = files.upload()
```

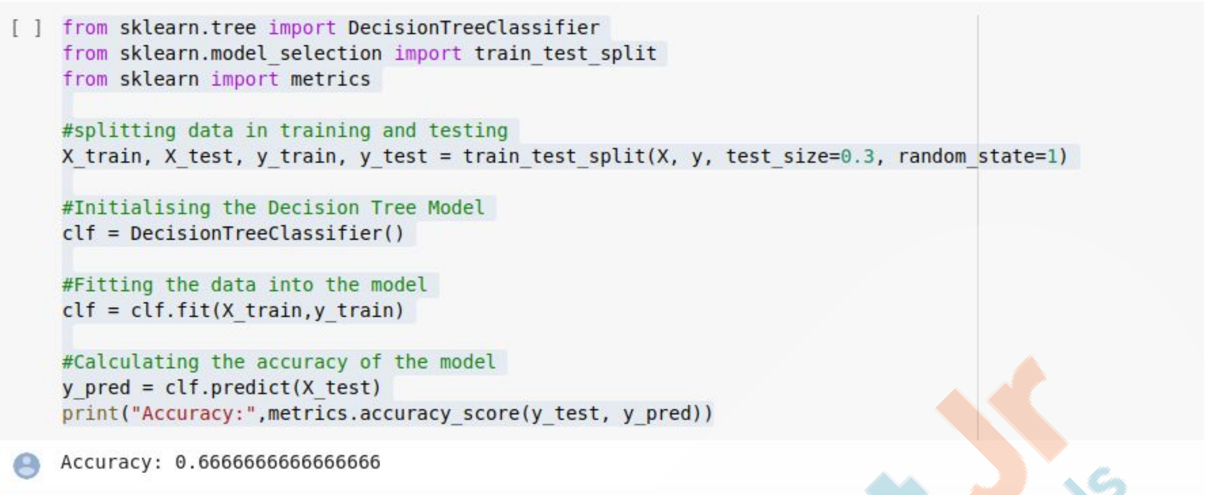
```
import pandas as pd

#Column Name
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']

df = pd.read_csv("diabetes.csv", names=col_names).iloc[1:]

print(df.head())
```

```
  pregnant glucose  bp skin insulin   bmi pedigree age label
1        6     148  72   35       0  33.6    0.627  50     1
2        1      85  66   29       0  26.6    0.351  31     0
3        8     183  64    0       0  23.3    0.672  32     1
4        1      89  66   23      94  28.1    0.167  21     0
5        0     137  40   35     168  43.1    2.288  33     1
```

```
features = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']
X = df[features]
y = df.label
```

	Now let's split the data to train and test and them fit the data in the model. Model fitting is a measure of how well a machine learning model generalizes to similar data to that on which it was trained. A model that is well-fitted produces more accurate outcomes. A model that is overfitted matches the data too closely. A model that is	Student helps the teacher with code for splitting the data in the model and then print the accuracy.

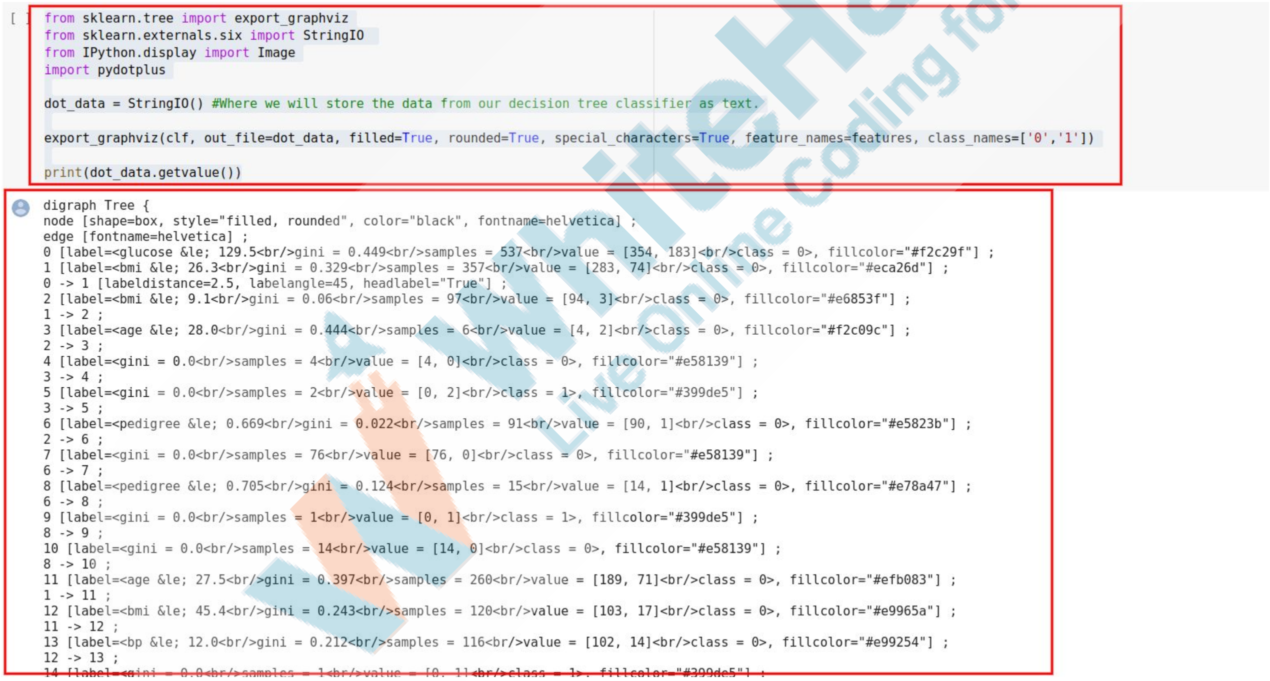
	underfitted doesn't match closely enough. <Teacher codes to split the data to train and test and then fit it in the model and then print the accuracy> Code:- **from sklearn.tree import DecisionTreeClassifier** **from sklearn.model_selection import train_test_split** **from sklearn import metrics** **#splitting data in training and testing** **X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)** **#Initialising the Decision Tree Model** **clf = DecisionTreeClassifier()** **#Fitting the data into the model** **clf = clf.fit(X_train,y_train)** **#Calculating the accuracy of the model** **y_pred = clf.predict(X_test)** **print("Accuracy:",metrics.accuracy_score(y_test, y_pred))**	



```
[ ] from sklearn.tree import DecisionTreeClassifier
    from sklearn.model_selection import train_test_split
    from sklearn import metrics

    #splitting data in training and testing
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

    #Initialising the Decision Tree Model
    clf = DecisionTreeClassifier()

    #Fitting the data into the model
    clf = clf.fit(X_train,y_train)

    #Calculating the accuracy of the model
    y_pred = clf.predict(X_test)
    print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

    Accuracy: 0.6666666666666666
```

	What is the accuracy we can see? Yes! so our model can predict if the person has diabetes with 0.66 accuracy. Now let's visualize this. To create a visualization for the Decision Tree Classifier we build above, we will use the **export_graphviz** module of python to first convert the data into text that we can read and understand, and then we'll use the pydotplus module to convert this text into an image. <Teacher codes to visualize the decision tree> Code:- **from sklearn.tree import export_graphviz** **from sklearn.externals.six import StringIO** **from IPython.display import Image** **import pydotplus**	ESR: we can see the accuracy of 0.66 Student observes and learns.

	dot_data = StringIO() #Where we will store the data from our decision tree classifier as text.
	export_graphviz(clf, out_file=dot_data, filled=True, rounded=True, special_characters=True, feature_names=features, class_names=['0','1'])
	print(dot_data.getvalue())	

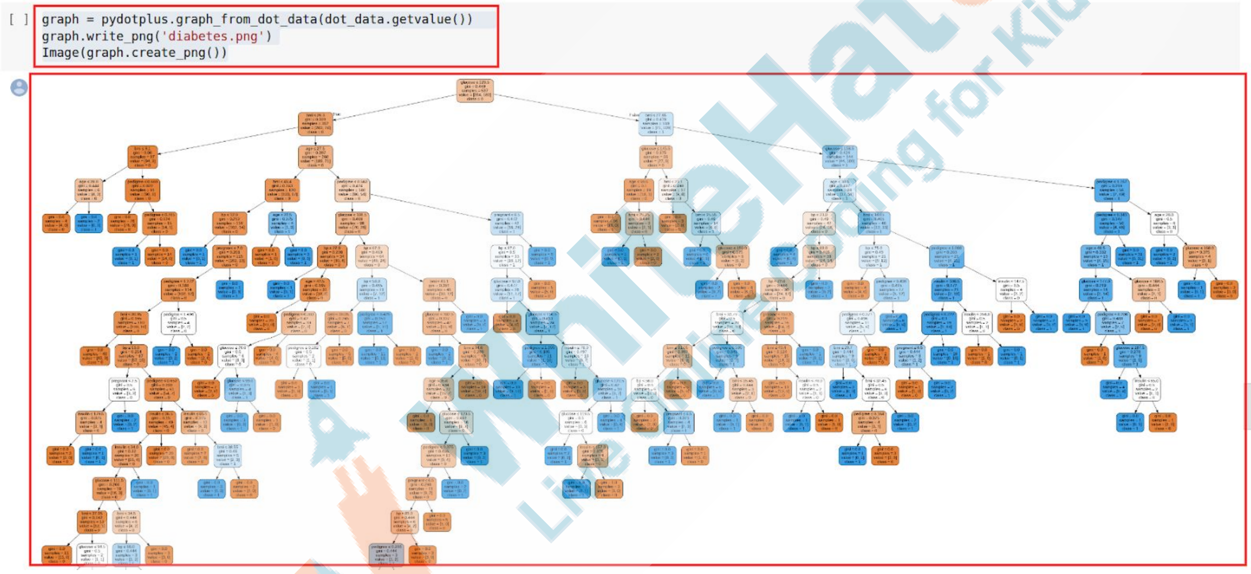


```
[ ] from sklearn.tree import export_graphviz
    from sklearn.externals.six import StringIO
    from IPython.display import Image
    import pydotplus

    dot_data = StringIO() #Where we will store the data from our decision tree classifier as text.

    export_graphviz(clf, out_file=dot_data, filled=True, rounded=True, special_characters=True, feature_names=features, class_names=['0','1'])

    print(dot_data.getvalue())
```

```
digraph Tree {
node [shape=box, style="filled, rounded", color="black", fontname=helvetica] ;
edge [fontname=helvetica] ;
0 [label=<glucose &le; 129.5<br/>gini = 0.449<br/>samples = 537<br/>value = [354, 183]<br/>class = 0>, fillcolor="#f2c29f"] ;
1 [label=<bmi &le; 26.3<br/>gini = 0.329<br/>samples = 357<br/>value = [283, 74]<br/>class = 0>, fillcolor="#eca26d"] ;
0 -> 1 [labeldistance=2.5, labelangle=45, headlabel="True"] ;
2 [label=<bmi &le; 9.1<br/>gini = 0.06<br/>samples = 97<br/>value = [94, 3]<br/>class = 0>, fillcolor="#e6853f"] ;
1 -> 2 ;
3 [label=<age &le; 28.0<br/>gini = 0.444<br/>samples = 6<br/>value = [4, 2]<br/>class = 0>, fillcolor="#f2c09c"] ;
2 -> 3 ;
4 [label=<gini = 0.0<br/>samples = 4<br/>value = [4, 0]<br/>class = 0>, fillcolor="#e58139"] ;
3 -> 4 ;
5 [label=<gini = 0.0<br/>samples = 2<br/>value = [0, 2]<br/>class = 1>, fillcolor="#399de5"] ;
3 -> 5 ;
6 [label=<pedigree &le; 0.669<br/>gini = 0.022<br/>samples = 91<br/>value = [90, 1]<br/>class = 0>, fillcolor="#e5823b"] ;
2 -> 6 ;
7 [label=<gini = 0.0<br/>samples = 76<br/>value = [76, 0]<br/>class = 0>, fillcolor="#e58139"] ;
6 -> 7 ;
8 [label=<pedigree &le; 0.705<br/>gini = 0.124<br/>samples = 15<br/>value = [14, 1]<br/>class = 0>, fillcolor="#e78a47"] ;
6 -> 8 ;
9 [label=<gini = 0.0<br/>samples = 1<br/>value = [0, 1]<br/>class = 1>, fillcolor="#399de5"] ;
8 -> 9 ;
10 [label=<gini = 0.0<br/>samples = 14<br/>value = [14, 0]<br/>class = 0>, fillcolor="#e58139"] ;
8 -> 10 ;
11 [label=<age &le; 27.5<br/>gini = 0.397<br/>samples = 260<br/>value = [189, 71]<br/>class = 0>, fillcolor="#efb083"] ;
1 -> 11 ;
12 [label=<bmi &le; 45.4<br/>gini = 0.243<br/>samples = 120<br/>value = [103, 17]<br/>class = 0>, fillcolor="#e9965a"] ;
11 -> 12 ;
13 [label=<bp &le; 12.0<br/>gini = 0.212<br/>samples = 116<br/>value = [102, 14]<br/>class = 0>, fillcolor="#e99254"] ;
12 -> 13 ;
14 [label=<gini = 0.0<br/>samples = 1<br/>value = [0, 1]<br/>class = 1>, fillcolor="#399de5"] ;
```

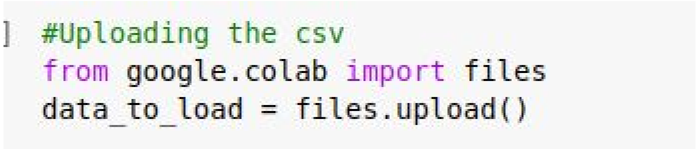
	Can you read what is printed ? Here we can see how our Decision Tree Classifier got converted into something that we can somewhat read and understand. Now, using the pydotplus, we will convert this into an	ESR: Varied!

<table>
<tr><td></td><td>image. Let's see how would that look like -

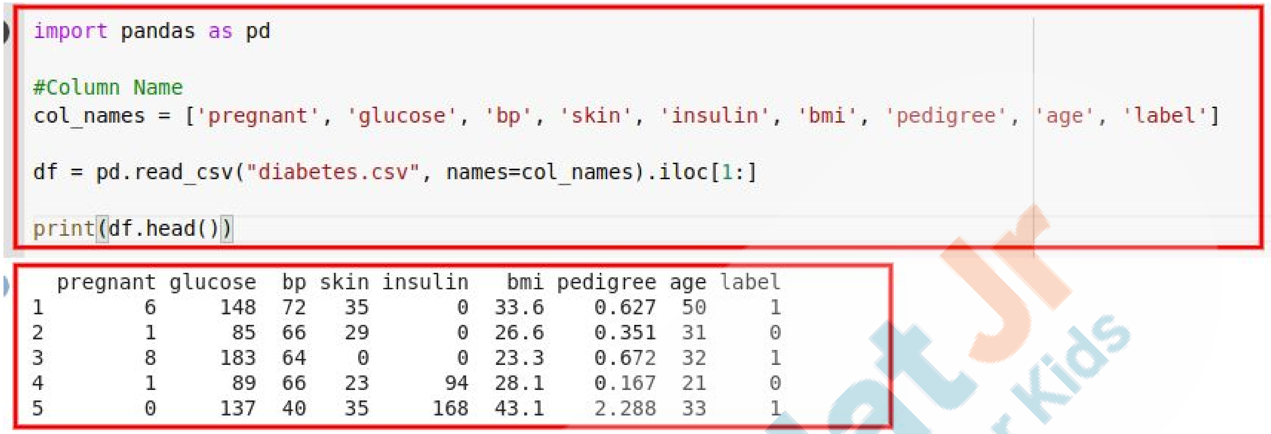
<Teacher codes to create a visualization of the plot>
Code:
<b>graph = pydotplus.graph_from_dot_data(dot_data.getvalue())</b>
<b>graph.write_png('diabetes.png')</b>
<b>Image(graph.create_png())</b></td><td></td></tr>
<tr><td colspan="3">
[ ] graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes.png')
Image(graph.create_png())</td></tr>
<tr><td></td><td>What can you see from the chart?

We can hardly make out anything, but each of the internal node has a decision rule using which, it splits the data.
From the chart above we can see that the chart goes much deeper from the root node.We can limit the max-depth of a Decision Tree Model as per our convenience.</td><td>ESR:
Varied!</td></tr>
</table>

	We can make the chart more understandable by doing some triming.	
	Can you try doing that? I'll help you wherever needed.	ESR: Yes!
Teacher Stops Screen Share		
	Now it's your turn. Please share your screen with me.	
● Ask Student to press ESC key to come back to panel **● Guide Student to start Screen Share** **● Teacher gets into Fullscreen**		
ACTIVITY **● Trim the data to make chart more understandable**		
Step 3: Student-Led Activity (15 min)	Teacher helps the student to open new Colab notebook and downlaod the data.	Student opens a new Colab Notebook from the Student Activity 1 Student downloads the data from Student Activity 2
	Teacher helps student to upload the data and create the data frames of it.	Student codes to upload the data and create the dataframes.

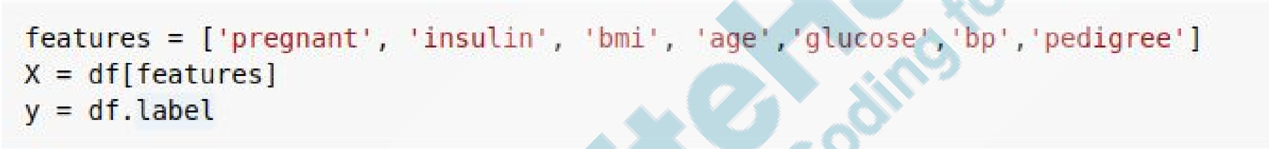


```
#Uploading the csv
from google.colab import files
data_to_load = files.upload()
```



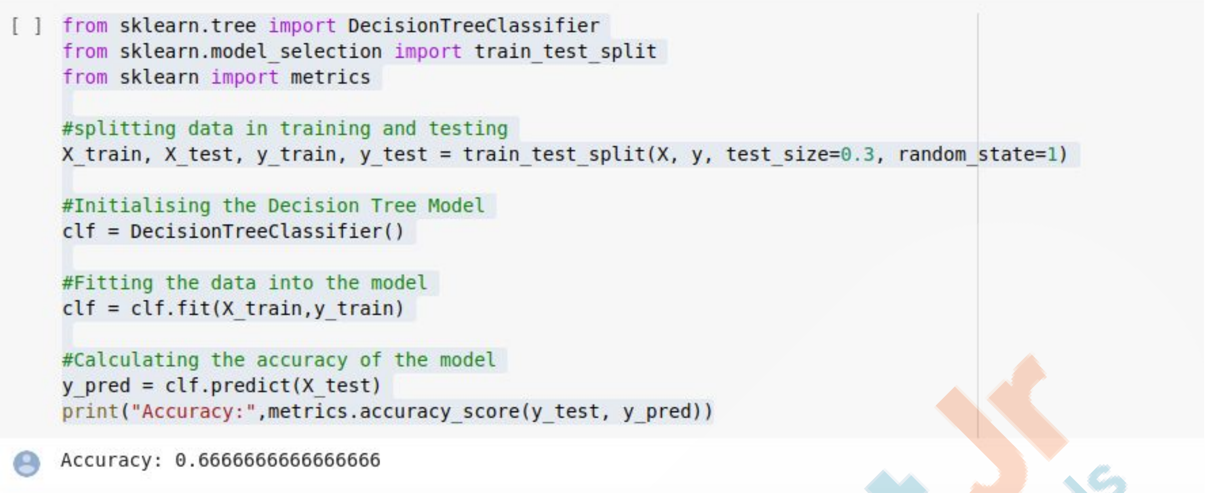
```
import pandas as pd

#Column Name
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']

df = pd.read_csv("diabetes.csv", names=col_names).iloc[1:]

print(df.head())
```

```
  pregnant glucose  bp skin insulin   bmi pedigree age label
1        6     148  72   35       0  33.6    0.627  50     1
2        1      85  66   29       0  26.6    0.351  31     0
3        8     183  64    0       0  23.3    0.672  32     1
4        1      89  66   23      94  28.1    0.167  21     0
5        0     137  40   35     168  43.1    2.288  33     1
```



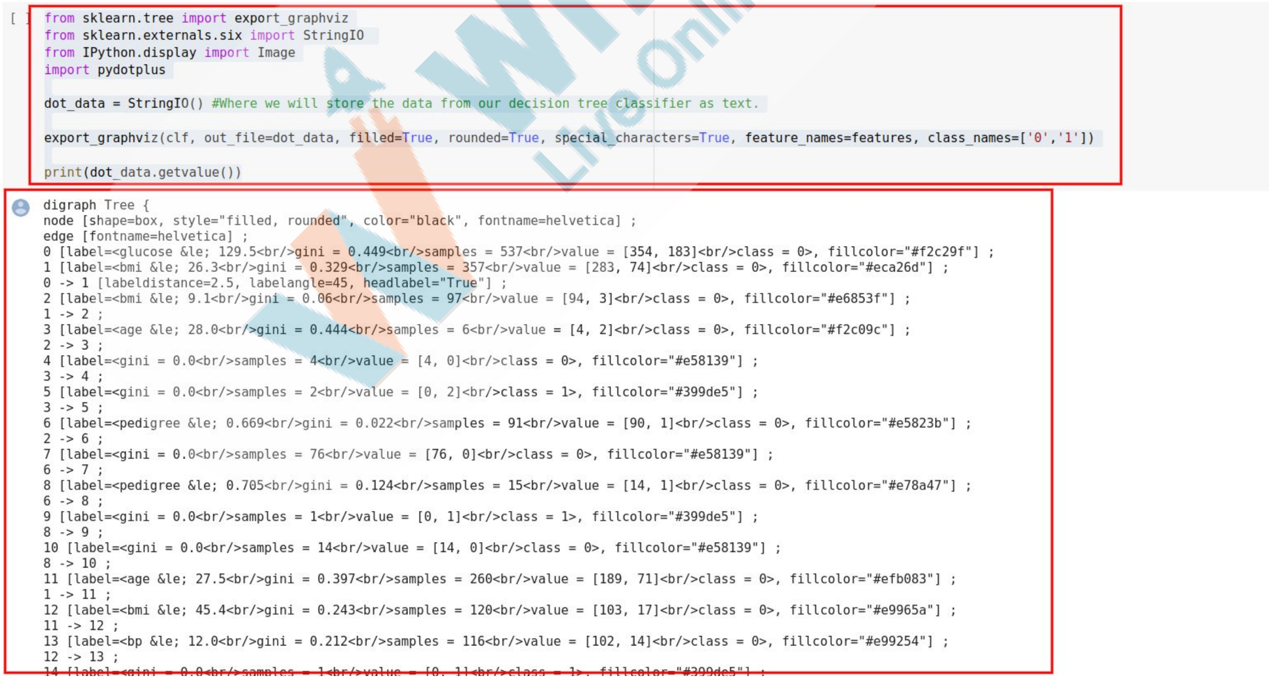
```
features = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']
X = df[features]
y = df.label
```

	Teacher helps the student to split the data to train , test and fit the model	student codes to split the data to train , test and fit the model



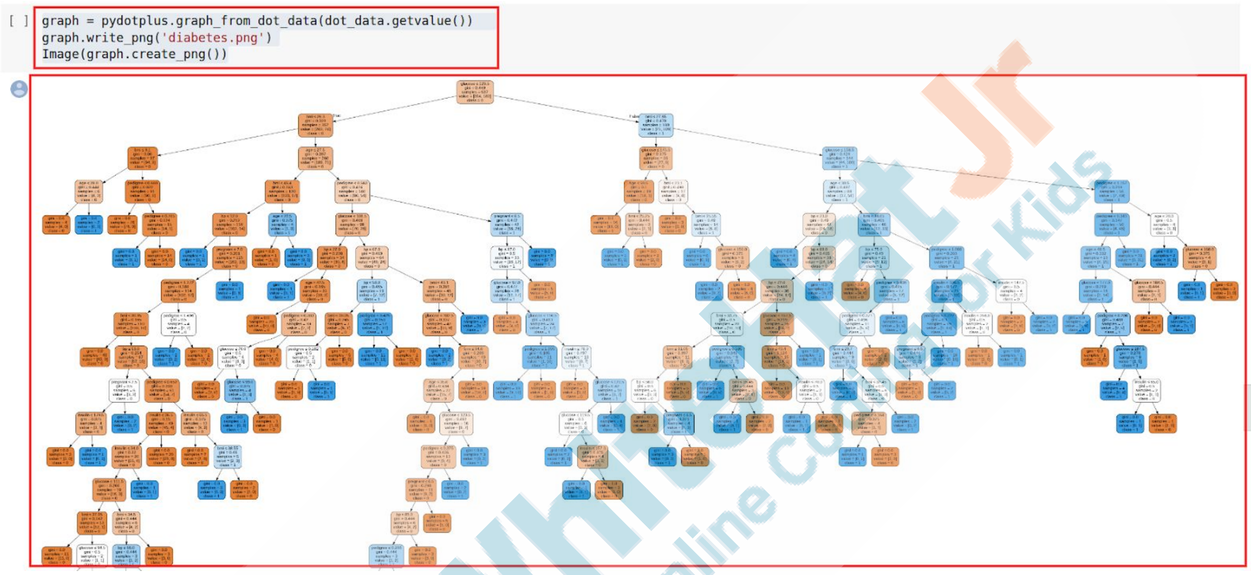
```
[ ] from sklearn.tree import DecisionTreeClassifier
    from sklearn.model_selection import train_test_split
    from sklearn import metrics

    #splitting data in training and testing
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

    #Initialising the Decision Tree Model
    clf = DecisionTreeClassifier()

    #Fitting the data into the model
    clf = clf.fit(X_train,y_train)

    #Calculating the accuracy of the model
    y_pred = clf.predict(X_test)
    print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

    Accuracy: 0.6666666666666666
```

	Teacher helps student to use the **export_graphviz** module of python to first convert the data into text that we can read and understand	Student codes to use **export_graphviz** module of python to first convert the data into text that we can read and understand



```
[ ] from sklearn.tree import export_graphviz
    from sklearn.externals.six import StringIO
    from IPython.display import Image
    import pydotplus

    dot_data = StringIO() #Where we will store the data from our decision tree classifier as text.

    export_graphviz(clf, out_file=dot_data, filled=True, rounded=True, special_characters=True, feature_names=features, class_names=['0','1'])

    print(dot_data.getvalue())

    digraph Tree {
    node [shape=box, style="filled, rounded", color="black", fontname=helvetica] ;
    edge [fontname=helvetica] ;
    0 [label=<glucose &le; 129.5<br/>gini = 0.449<br/>samples = 537<br/>value = [354, 183]<br/>class = 0>, fillcolor="#f2c29f"] ;
    1 [label=<bmi &le; 26.3<br/>gini = 0.329<br/>samples = 357<br/>value = [283, 74]<br/>class = 0>, fillcolor="#eca26d"] ;
    0 -> 1 [labeldistance=2.5, labelangle=45, headlabel="True"] ;
    2 [label=<bmi &le; 9.1<br/>gini = 0.06<br/>samples = 97<br/>value = [94, 3]<br/>class = 0>, fillcolor="#e6853f"] ;
    1 -> 2 ;
    3 [label=<age &le; 28.0<br/>gini = 0.444<br/>samples = 6<br/>value = [4, 2]<br/>class = 0>, fillcolor="#f2c09c"] ;
    2 -> 3 ;
    4 [label=<gini = 0.0<br/>samples = 4<br/>value = [4, 0]<br/>class = 0>, fillcolor="#e58139"] ;
    3 -> 4 ;
    5 [label=<gini = 0.0<br/>samples = 2<br/>value = [0, 2]<br/>class = 1>, fillcolor="#399de5"] ;
    3 -> 5 ;
    6 [label=<pedigree &le; 0.669<br/>gini = 0.022<br/>samples = 91<br/>value = [90, 1]<br/>class = 0>, fillcolor="#e5823b"] ;
    2 -> 6 ;
    7 [label=<gini = 0.0<br/>samples = 76<br/>value = [76, 0]<br/>class = 0>, fillcolor="#e58139"] ;
    6 -> 7 ;
    8 [label=<pedigree &le; 0.705<br/>gini = 0.124<br/>samples = 15<br/>value = [14, 1]<br/>class = 0>, fillcolor="#e78a47"] ;
    6 -> 8 ;
    9 [label=<gini = 0.0<br/>samples = 1<br/>value = [0, 1]<br/>class = 1>, fillcolor="#399de5"] ;
    8 -> 9 ;
    10 [label=<gini = 0.0<br/>samples = 14<br/>value = [14, 0]<br/>class = 0>, fillcolor="#e58139"] ;
    8 -> 10 ;
    11 [label=<age &le; 27.5<br/>gini = 0.397<br/>samples = 260<br/>value = [189, 71]<br/>class = 0>, fillcolor="#efb083"] ;
    1 -> 11 ;
    12 [label=<bmi &le; 45.4<br/>gini = 0.243<br/>samples = 120<br/>value = [103, 17]<br/>class = 0>, fillcolor="#e9965a"] ;
    11 -> 12 ;
    13 [label=<bp &le; 12.0<br/>gini = 0.212<br/>samples = 116<br/>value = [102, 14]<br/>class = 0>, fillcolor="#e99254"] ;
    12 -> 13 ;
    14 [label=<gini = 0.0<br/>samples = 1<br/>value = [0, 1]<br/>class = 1>, fillcolor="#399de5"] ;
```

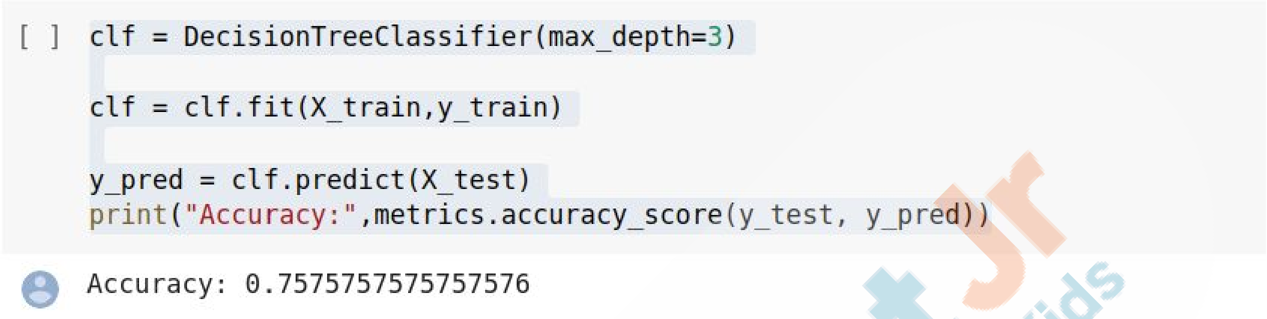
	Teacher helps the student to convert this text into image using the **pydotplus** module	Student codes to convert the text into image by using the **pydotplus** module.



```
[ ] graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
    graph.write_png('diabetes.png')
    Image(graph.create_png())
```

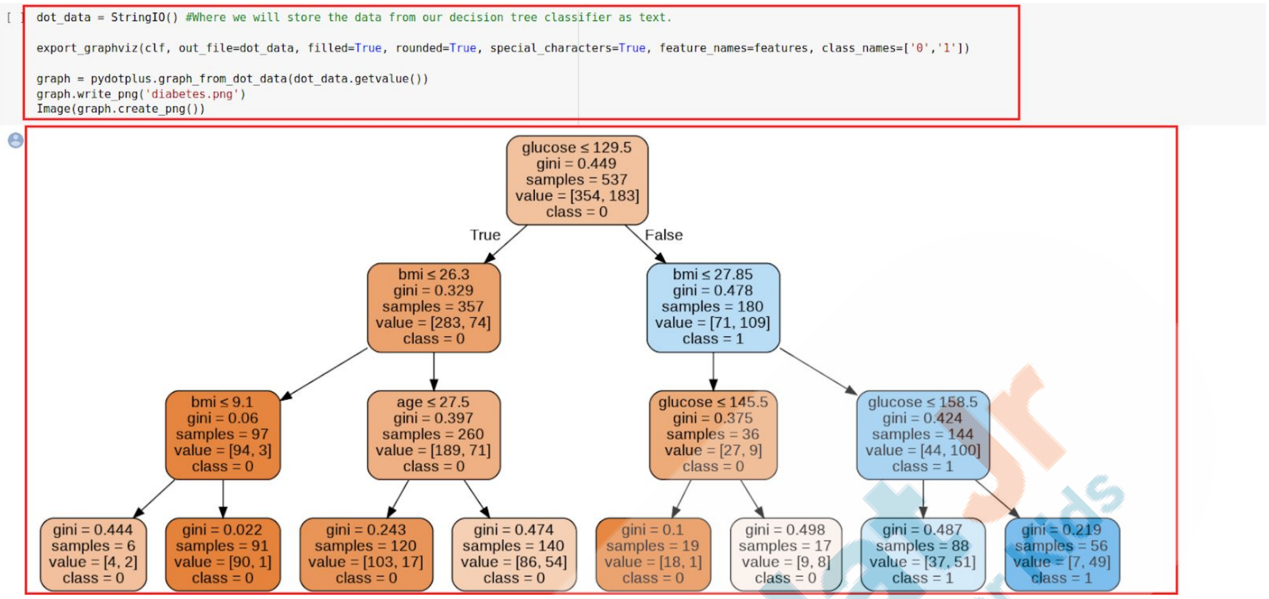
	So now we are going to trim the chart so that we can make it more understandable. And we can do that by just providing the **max_depth** value to the **DecisionTreeClassifier** module. Teacher helps the student with the code. Code: **clf = DecisionTreeClassifier(max_depth =3)** **clf = clf.fit(X_train,y_train)**	Student codes to pass the value of **max_depth =3** to the **DecisionTreeClassifier**

	y_pred = clf.predict(X_test) **print("Accuracy:",metrics.accuracy_score(y_test, y_pred))**	



```
[ ] clf = DecisionTreeClassifier(max_depth=3)

    clf = clf.fit(X_train,y_train)

    y_pred = clf.predict(X_test)
    print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

    Accuracy: 0.7575757575757576
```

	Now let's create a visualization of this trimmed data. Teacher helps student to code for the same. Code: **dot_data = StringIO() #Where we will store the data from our decision tree classifier as text.** **export_graphviz(clf, out_file=dot_data, filled=True, rounded=True, special_characters=True, feature_names=features, class_names=['0','1'])** **graph = pydotplus.graph_from_dot_data(dot_data.getvalue())** **graph.write_png('diabetes.png')** **Image(graph.create_png())**	Student codes to create this data into image.



```
[ ] dot_data = StringIO() #Where we will store the data from our decision tree classifier as text.

    export_graphviz(clf, out_file=dot_data, filled=True, rounded=True, special_characters=True, feature_names=features, class_names=['0','1'])

    graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
    graph.write_png('diabetes.png')
    Image(graph.create_png())
```

	Here, we can see that the tree is much more readable and understandable. We set the max-depth to 3, so it only goes 3 layers down from the root node. And by looking at the chart what can we conclude?	ESR: By looking at this chart, we can say with almost 75% accuracy that a person who's **Glucose** is greater than 129.5 and, **BMI** is greater than 27.85 Is more prone to be a Diabetes Patient.
	Yes perfect.	

Teacher Guides Student to Stop Screen Share

FEEDBACK

- **Appreciate the student for their efforts**

● **Identify 2 strengths and 1 area of progress for the student**		
Step 4: **Wrap-Up** **(5 min)**	Now let's quickly go through what we did today?	ESR: we split the data to train, test and fit the data into the model. We converted the data into an image of charts. We trimmed the charts for better understanding.
	Awesome. You can use other data and practise for this model for better understanding. In the next class we'll explore more of machine learning. See you then	-
Teacher Clicks ✖ End Class		
Additional Activities	*Encourage the student to write reflection notes in their reflection journal using markdown.* Use these as guiding questions: ● What happened today? - Describe what happened - Code I wrote ● How did I feel after the class? ● What have I learned about programming and developing games?	*The student uses the markdown editor to write her/his reflection in a reflection journal.*

	- What aspects of the class helped me? What did I find difficult?	

Activity	Activity Name	Links
Teacher Activity 1	Google Colab notebook	https://colab.research.google.com/notebooks/intro.ipynb#recent=true
Teacher Activity 2	diabetes data	https://raw.githubusercontent.com/whitehatjr/datasets/master/C119/diabetes.csv
Teacher Activity 3	Solution	https://colab.research.google.com/drive/1EYKK1VMxAJpZ88-kWhm6_5kXklJZAmWP?usp=sharing
Student Activity 1	Google Colab notebook	https://colab.research.google.com/notebooks/intro.ipynb#recent=true
Student Activity 2	diabetes data	https://raw.githubusercontent.com/whitehatjr/datasets/master/C119/diabetes.csv