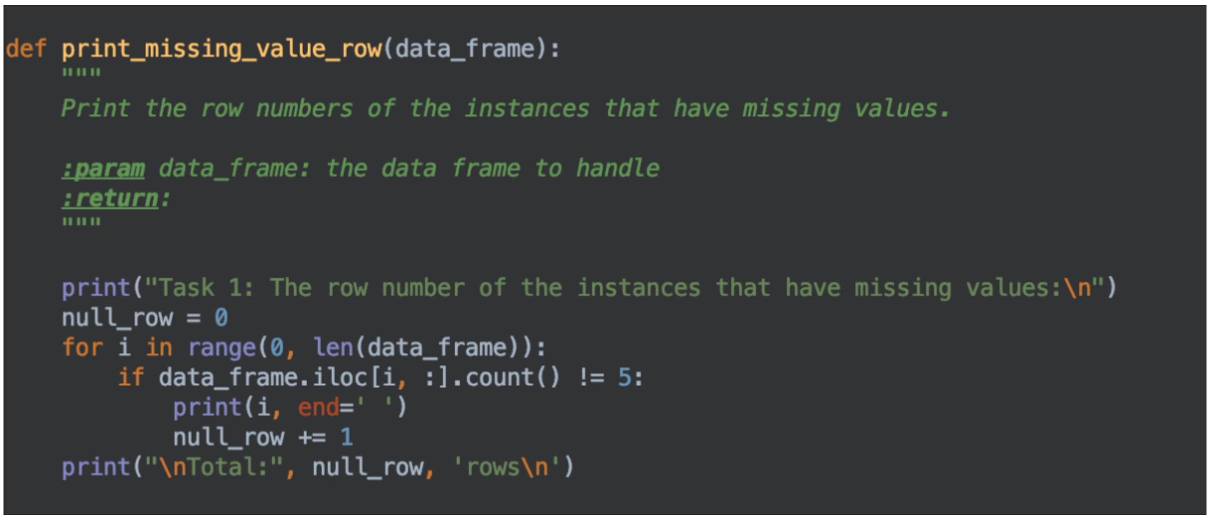
Major	Information and Computer Science
Module Code	CSE 315
Assignment Code	Assignment 1
Student ID	1509264
Name	ZHIYONG LIU

Assignment 1 Report

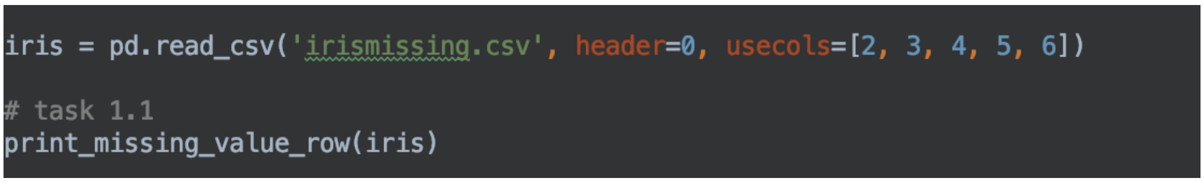
Task1 Data Pre-processing:

1. Import the dataset irisMissing.csv into a data frame and discover the row numbers of the instances that have missing values.

   Code:

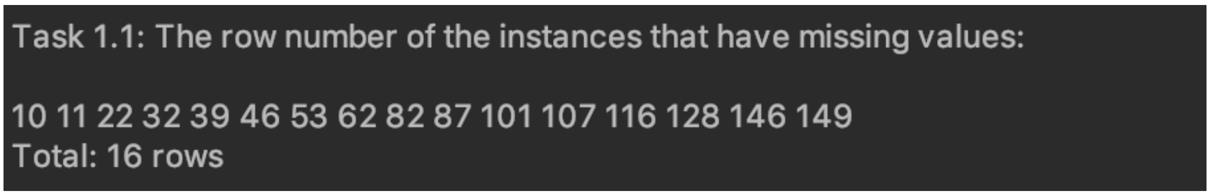


```python
def print_missing_value_row(data_frame):
    """
    Print the row numbers of the instances that have missing values.

    :param data_frame: the data frame to handle
    :return:
    """

    print("Task 1: The row number of the instances that have missing values:\n")
    null_row = 0
    for i in range(0, len(data_frame)):
        if data_frame.iloc[i, :].count() != 5:
            print(i, end=' ')
            null_row += 1
    print("\nTotal:", null_row, 'rows\n')
```



```python
iris = pd.read_csv('irismissing.csv', header=0, usecols=[2, 3, 4, 5, 6])

# task 1.1
print_missing_value_row(iris)
```

   Output:



```
Task 1.1: The row number of the instances that have missing values:

10 11 22 32 39 46 53 62 82 87 101 107 116 128 146 149
Total: 16 rows
```

It shows that the irismissing.csv has 16 rows that has missing values, the row numbers are shown in output.

2. Write a program to drop missing values, and describe three other strategies (median, mean and mode) for handling missing values and write a function to implement of these strategies.

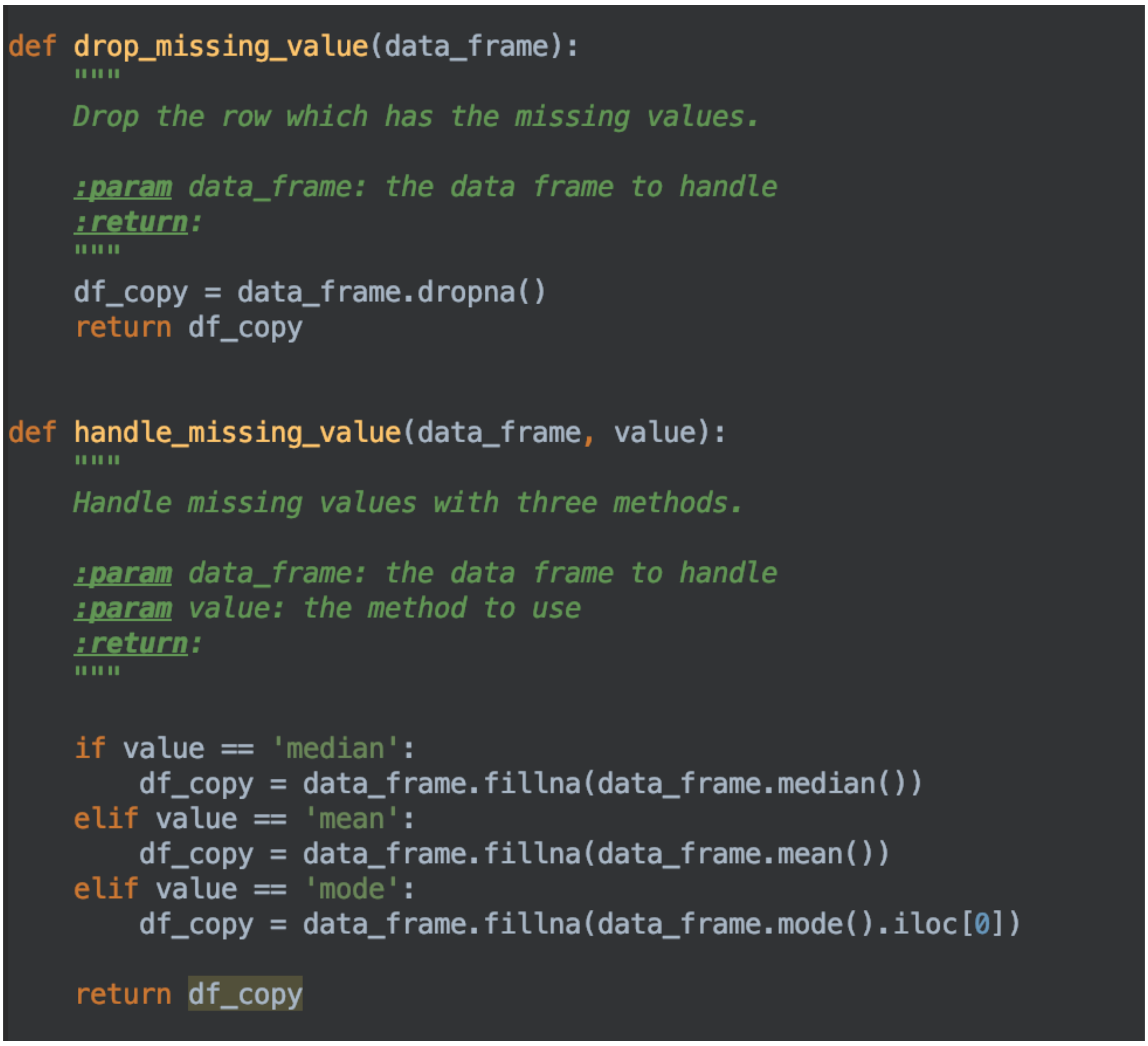
Drop: drop the row that has the missing value.

Median: use the median value to fill the missing value.

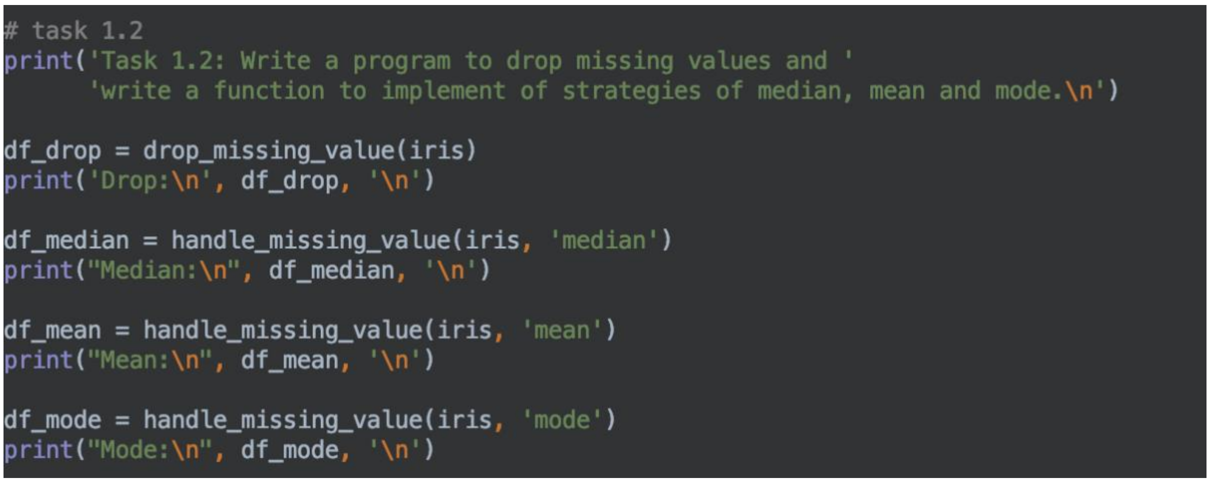
Mean: use the mean value to fill the missing value.

Mode: use the mode value to fill the missing value.

Code:



```python
def drop_missing_value(data_frame):
    """
    Drop the row which has the missing values.

    :param data_frame: the data frame to handle
    :return:
    """
    df_copy = data_frame.dropna()
    return df_copy


def handle_missing_value(data_frame, value):
    """
    Handle missing values with three methods.

    :param data_frame: the data frame to handle
    :param value: the method to use
    :return:
    """

    if value == 'median':
        df_copy = data_frame.fillna(data_frame.median())
    elif value == 'mean':
        df_copy = data_frame.fillna(data_frame.mean())
    elif value == 'mode':
        df_copy = data_frame.fillna(data_frame.mode().iloc[0])

    return df_copy
```



```python
# task 1.2
print('Task 1.2: Write a program to drop missing values and '
      'write a function to implement of strategies of median, mean and mode.\n')

df_drop = drop_missing_value(iris)
print('Drop:\n', df_drop, '\n')

df_median = handle_missing_value(iris, 'median')
print("Median:\n", df_median, '\n')

df_mean = handle_missing_value(iris, 'mean')
print("Mean:\n", df_mean, '\n')

df_mode = handle_missing_value(iris, 'mode')
print("Mode:\n", df_mode, '\n')
```

Output:

Drop: [134 rows x 5 columns]

Median: [150 rows x 5 columns]

Mean: [150 rows x 5 columns]

Mode: [150 rows x 5 columns]

To see the detailed information, run the program.

3. Compare the results of applying each missing value strategy using some visualization method.

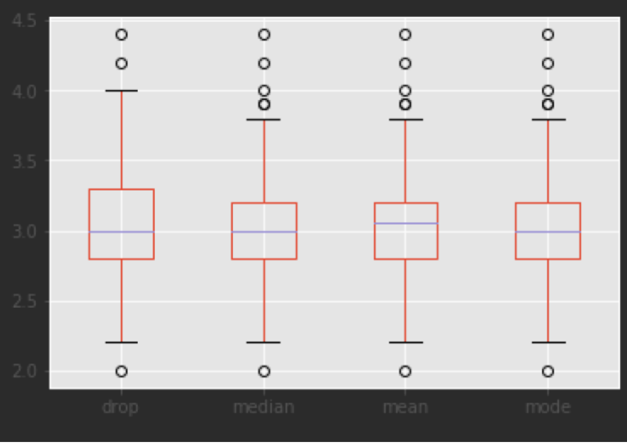
Use the box plot to compare each missing value strategy:

Code:

```
# task 1.3
print('Task 1.3: Compare the results of applying each missing value strategy '
      'using some visualization method')
df = pd.DataFrame({'drop': df_drop['Sepal.Width'],
                   'median': df_median['Sepal.Width'],
                   'mean': df_mean['Sepal.Width'],
                   'mode': df_mode['Sepal.Width']})
df.plot.box()
```

Output:

```
Task 1.3: Compare the results of applying each missing value strategy using some visualization method

Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x11e31afd0>
```



Analyze:

Drop method
maximum: 4.0
third quartile: 3.3
median: 3.0
first quartile: 2.8
minimum:2.2

Median method
maximum: 4.0
third quartile: 3.2
median: 3.0
first quartile: 2.8
minimum: 2.2

Mean method
maximum: 4.0
third quartile: 3.2
median: 3.06
first quartile: 2.8
minimum: 2.2

Mode method
maximum: 4.0
third quartile: 3.2
median: 3.0
first quartile: 2.8
minimum: 2.2

maximum:
drop = median = mean = mode

third quartile:
drop > median = mean = mode

median:
mean > drop = median =mode

first quartile:
drop = median = mean = mode

minimum:
drop = median = mean = mode

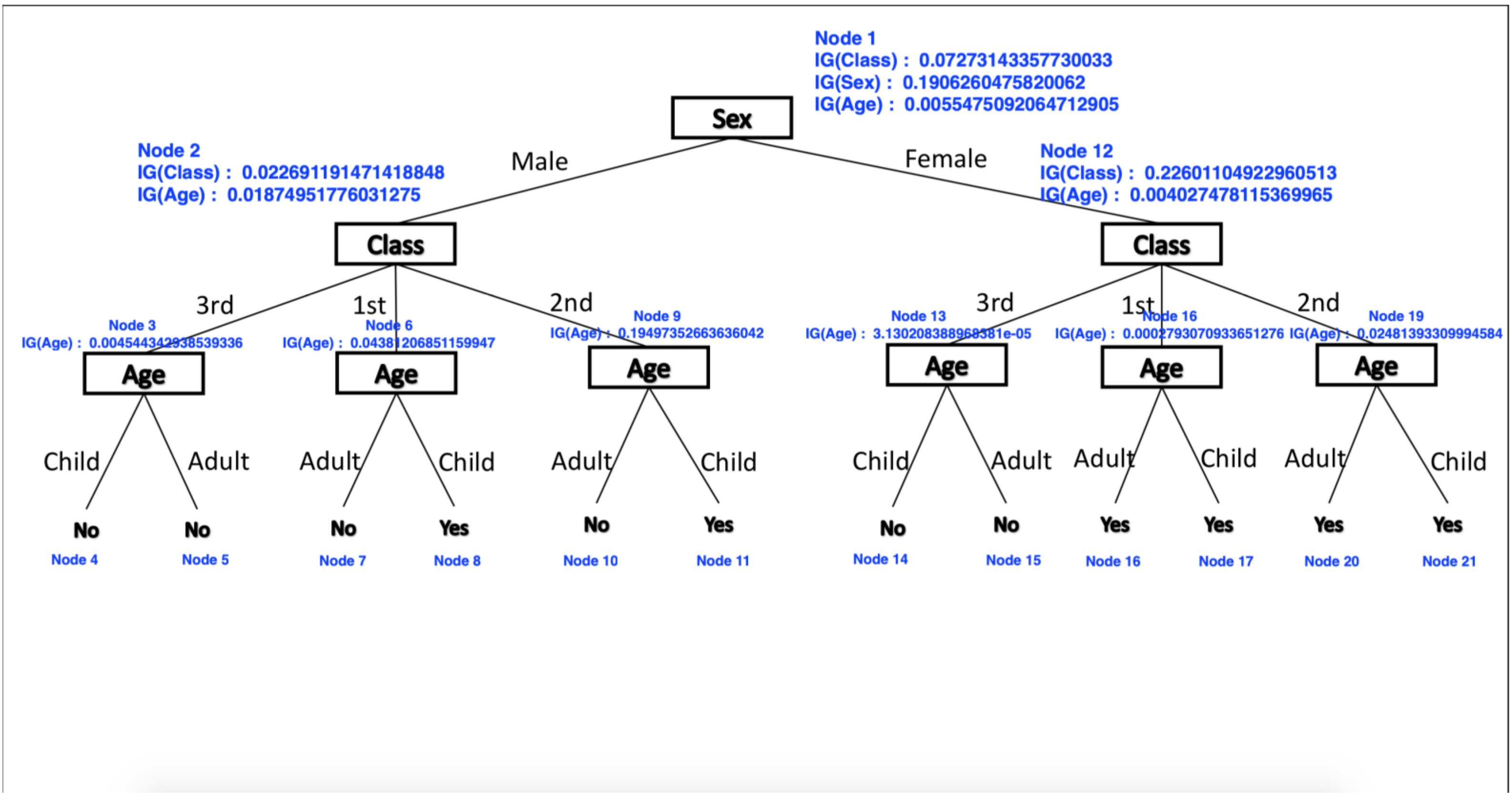
Standard Deviation:

drop	0.417459
median	0.394864
mean	0.394409
mode	0.394864

Task 2 Decision Trees

1. **Manually generate the decision tree for the Titanic passenger survival dataset below. Use Information Gain as your split measure.**

   The output decision tree:



Node 1
IG(Class) : 0.07273143357730033
IG(Sex) : 0.1906260475820062
IG(Age) : 0.0055475092064712905
Sex
Male
Female
Node 2
IG(Class) : 0.022691191471418848
IG(Age) : 0.01874951776031275
Node 12
IG(Class) : 0.22601104922960513
IG(Age) : 0.004027478115369965
Class
Class
3rd
1st
2nd
3rd
1st
2nd
Node 3
IG(Age) : 0.004544342938539336
Node 6
IG(Age) : 0.04381206851159947
Node 9
IG(Age) : 0.19497352663636042
Node 13
IG(Age) : 3.130208388968381e-05
Node 16
IG(Age) : 0.0002793070933651276
Node 19
IG(Age) : 0.02481393309994584
Age
Age
Age
Age
Age
Age
Child
Adult
Adult
Child
Adult
Child
Child
Adult
Adult
Child
Adult
Child
No
No
No
Yes
No
Yes
No
No
Yes
Yes
Yes
Yes
Node 4
Node 5
Node 7
Node 8
Node 10
Node 11
Node 14
Node 15
Node 16
Node 17
Node 20
Node 21

In the plot I label the information gain in each node which helps to show if the decision tree is right.

The process of generate:

1. Node 1

Because all the observations are not same class and there are three attributes remained (sex, class, age), so we need to choose one attribute as the root node.
The information gain of all the attributes:
IG(Class): 0.07273143357730033
IG(Sex): 0.1906260475820062
IG(Age): 0.0055475092064712905
Because the information gain of sex is largest, so choose the sex as the root node.

We split the dataset by the value of attribute sex, the left branch is dataset with the value of Male, the right branch is dataset with the value of Female.

2. Node 2
The set S_Male is not empty
The observations of set S_Male are not same class.
The remained attributes: Class, Age

So we need to choose one attribute as the node 2.
The information gain of all the attributes:
IG(Class): 0.022691191471418848
IG(Age): 0.01874951776031275
Because the information gain of class is largest, so choose the class as the node 2.

We split the dataset by the value of attribute class, the left branch is dataset with the value of 3rd, the middle branch is dataset with the value of 1 st, the right branch is dataset with the value of 2nd.

3. Node 3
The set S_3rd is not empty
The observations of S_3rd are not same class.
The remained attributes: age
So choose the age as the node 3.

We split the dataset by the value of attribute age, the left branch is dataset with the value of Child, the right branch is dataset with the value of Adult.

4. Node 4
The set S_Child is not empty
The observation of S_Child are not same class
The remained attributes: no attributes
So return the most common value:
No: 35, Yes: 13

So Node 4: No

5. Node 5
The set S_Adult is not empty
The observations of S_Child are not same class
The remained attributes: no attributes
So return the most common value:
No: 387, Yes: 75

So Node 5: No

6. Node 6
The set S_1st is not empty
The observations of S_1st are not same class.

The remained attributes: age
So choose the age as the node 6.

We split the dataset by the value of attribute age, the left branch is dataset with the value of Adult, the right branch is dataset with the value of Child.

7. Node 7
The set S_Adult is not empty
The observations of S_Adult are not same class.
The remained attributes: no attributes
So return the most common value:
No: 118, Yes: 57

So Node 7: No

8. Node 8
The set S_Child is not empty
The observations of S_Child are same class Yes (all 5 observations are class Yes)

So Node 4: Yes

9. Node 9
The set S_2nd is not empty
The observations of S_2nd are not same class.
The remained attributes: age
So choose the age as the node 9.

We split the dataset by the value of attribute age, the left branch is dataset with the value of Adult, the right branch is dataset with the value of Child.

10. Node 10
The set S_Adult is not empty
The observations of S_Adult are not same class.
The remained attributes: no attributes
So return the most common value:
No: 154, Yes: 14

So Node 10: No

11. Node 11
The set S_Child is not empty
The observations of S_Child are same class(all 11 observations are class Yes)

So Node 11: Yes

12. Node 12
The set S_Female is not empty
The observations of set S_Female are not same class.
The remained attributes: Class, Age

So we need to choose one attribute as the node 12.
The information gain of all the attributes:
IG(Class): 0.22601104922960513
IG(Age): 0.004027478115369965
Because the information gain of class is largest, so choose the class as the node 12.

We split the dataset by the value of attribute class, the left branch is dataset with the value of 3rd, the middle branch is dataset with the value of 1 st, the right branch is dataset with the value of 2nd.

13. Node 13
The set S_3rd is not empty
The observations of set S_3rd are not same class.
The remained attributes: Age
So choose the age as the node 13.

We split the dataset by the value of attribute age, the left branch is dataset with the value of Child, the right branch is dataset with the value of Adult.

14. Node 14
The set S_Child is not empty
The observations of set S_Child are not same class.
The remained attributes: no attributes
So return the most common value:
No: 17, Yes: 14

So Node 14: No

15. Node 15
The set S_Adult is not empty
The observations of set S_Adult are not same class.
The remained attributes: no attributes
So return the most common value:
No: 89, Yes: 76

So Node 15: No

16. Node 16
The set S_1st is not empty
The observations of set S_1st are not same class.

The remained attributes: Age
So choose the age as the node 16.

We split the dataset by the value of attribute age, the left branch is dataset with the value of Adult, the right branch is dataset with the value of Child.

17. Node 17
The set S_Adult is not empty
The observations of set S_Adult are not same class.
The remained attributes: no attributes
So return the most common value:
No: 4, Yes: 140

So Node 17: Yes

18. Node 18
The set S_Child is not empty
The observations of set S_Child are same class (all one observation are class Yes)

So Node 18: Yes

19. Node 19
The set S_2nd is not empty
The observations of set S_2nd are not same class.
The remained attributes: Age
So choose the age as the node 19.

We split the dataset by the value of attribute age, the left branch is dataset with the value of Adult, the right branch is dataset with the value of Child.

20. Node 20
The set S_Adult is not empty
The observations of set S_Adult are not same class.
The remained attributes: no attributes
So return the most common value:
No: 13, Yes: 80

So Node 20: Yes

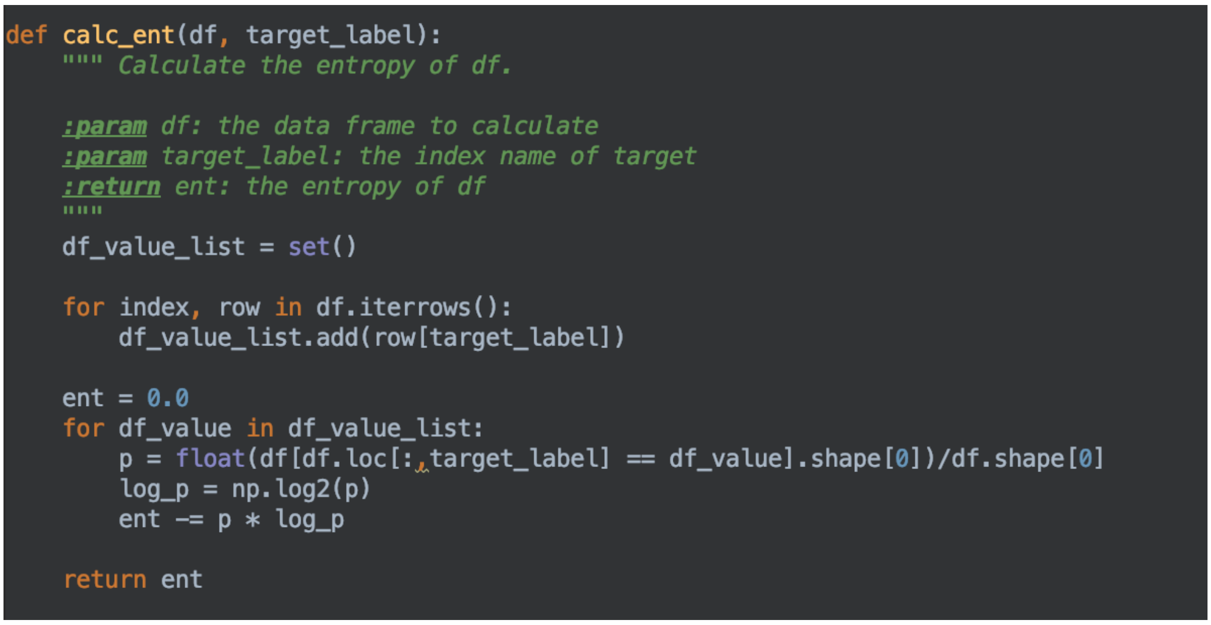
21. Node 21
The set S_Child is not empty
The observations of set S_Child are same class(all 13 observations are class Yes).

So Node 21: Yes

2. **In the figure you can find the Tennis data from the example we used at Lecture on describing the ID3 algorithm.**

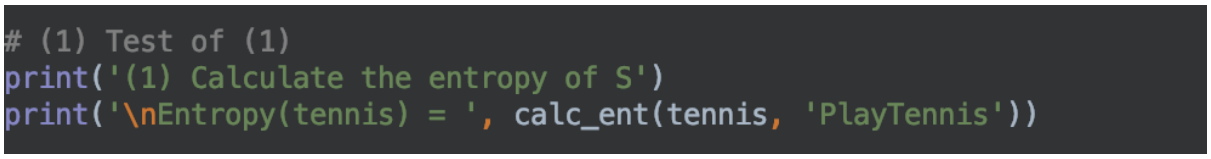
(1) Write a function that computes the Entropy of a set S with N_{pos} positive observations and N_{neg} negative observations.

Code:



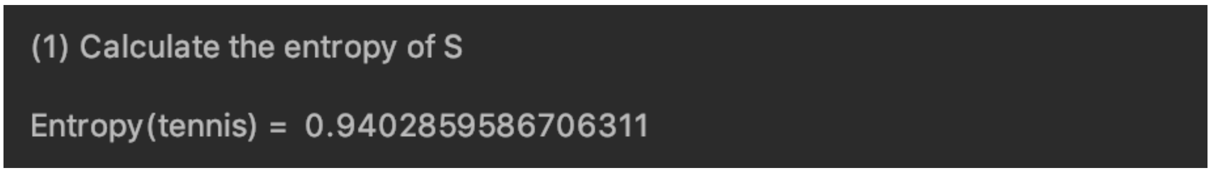
```python
def calc_ent(df, target_label):
    """ Calculate the entropy of df.

    :param df: the data frame to calculate
    :param target_label: the index name of target
    :return ent: the entropy of df
    """
    df_value_list = set()

    for index, row in df.iterrows():
        df_value_list.add(row[target_label])

    ent = 0.0
    for df_value in df_value_list:
        p = float(df[df.loc[:,target_label] == df_value].shape[0])/df.shape[0]
        log_p = np.log2(p)
        ent -= p * log_p

    return ent
```

Test:



```python
# (1) Test of (1)
print('(1) Calculate the entropy of S')
print('\nEntropy(tennis) = ', calc_ent(tennis, 'PlayTennis'))
```

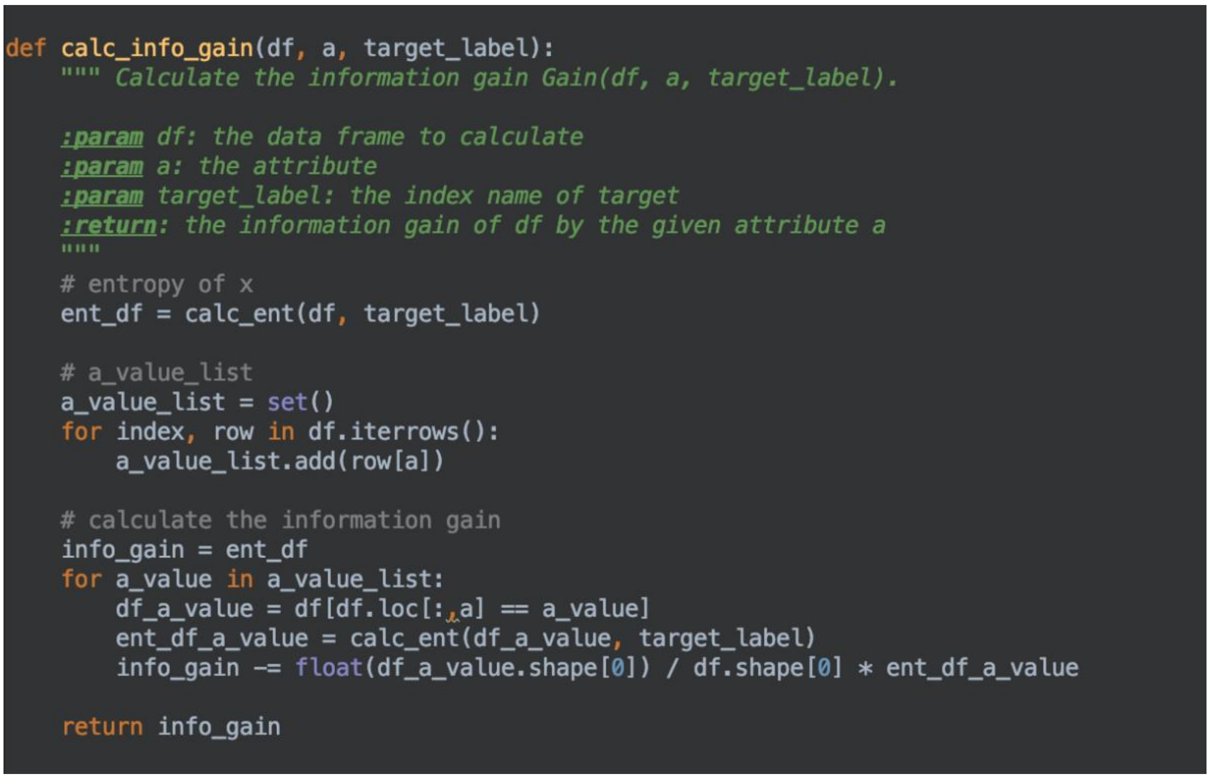
Output:



```
(1) Calculate the entropy of S

Entropy(tennis) =  0.9402859586706311
```

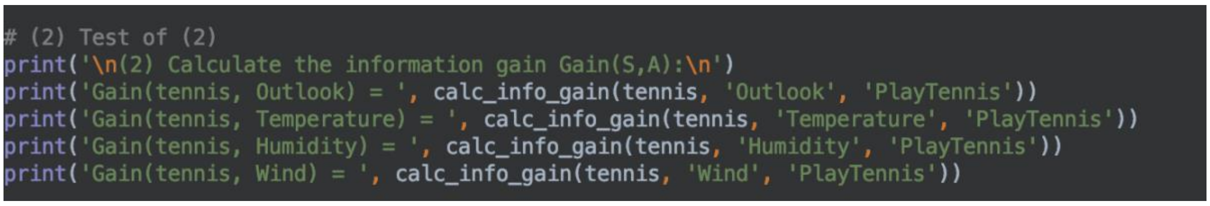
(2) Write a function that takes as input a set S of observations and an attribute A from these observations, and calculates the Information Gain, denoted as Gain (S, A), as if we were to split on that attribute in the context of the ID3 decision tree algorithm.

Code:



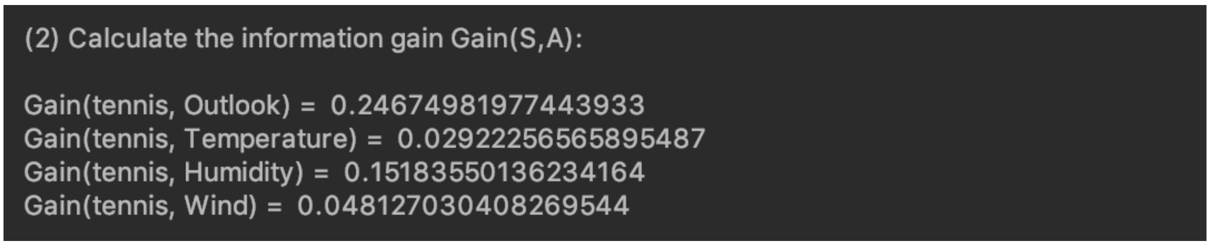
```python
def calc_info_gain(df, a, target_label):
    """ Calculate the information gain Gain(df, a, target_label).

    :param df: the data frame to calculate
    :param a: the attribute
    :param target_label: the index name of target
    :return: the information gain of df by the given attribute a
    """
    # entropy of x
    ent_df = calc_ent(df, target_label)

    # a_value_list
    a_value_list = set()
    for index, row in df.iterrows():
        a_value_list.add(row[a])

    # calculate the information gain
    info_gain = ent_df
    for a_value in a_value_list:
        df_a_value = df[df.loc[:,a] == a_value]
        ent_df_a_value = calc_ent(df_a_value, target_label)
        info_gain -= float(df_a_value.shape[0]) / df.shape[0] * ent_df_a_value

    return info_gain
```

Test:



```python
# (2) Test of (2)
print('\n(2) Calculate the information gain Gain(S,A):\n')
print('Gain(tennis, Outlook) = ', calc_info_gain(tennis, 'Outlook', 'PlayTennis'))
print('Gain(tennis, Temperature) = ', calc_info_gain(tennis, 'Temperature', 'PlayTennis'))
print('Gain(tennis, Humidity) = ', calc_info_gain(tennis, 'Humidity', 'PlayTennis'))
print('Gain(tennis, Wind) = ', calc_info_gain(tennis, 'Wind', 'PlayTennis'))
```

Output:

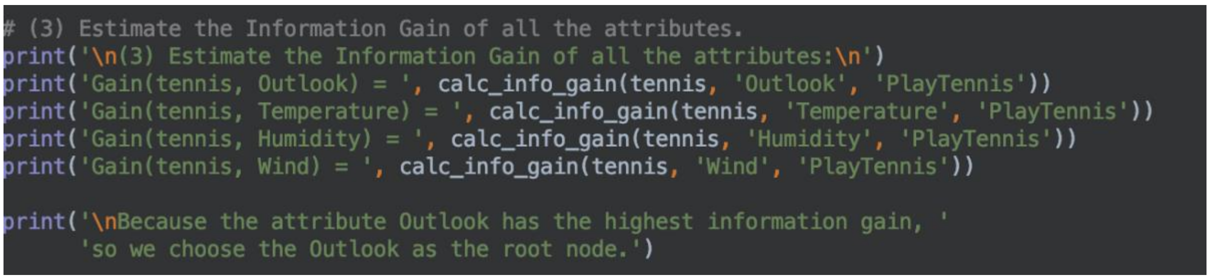


```
(2) Calculate the information gain Gain(S,A):

Gain(tennis, Outlook) = 0.24674981977443933
Gain(tennis, Temperature) = 0.029222565658955487
Gain(tennis, Humidity) = 0.15183550136234164
Gain(tennis, Wind) = 0.04812703040826544
```

(3) Estimate the Information Gain of all the attributes. Which one would you choose for the root node of your decision tree?

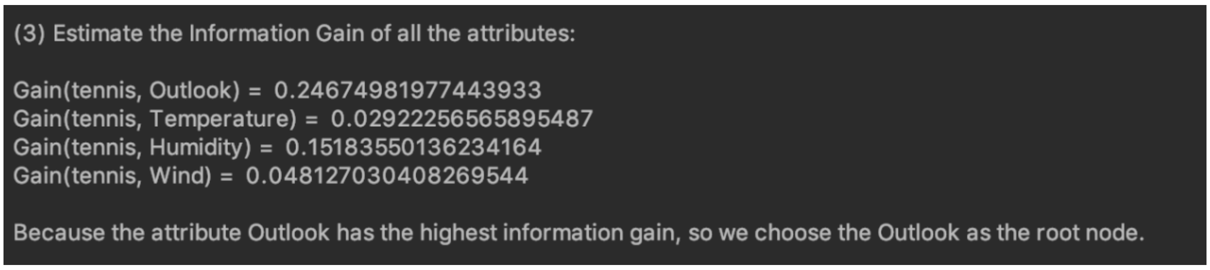
Use the method in (2), we can choose the best attribute which has the largest information gain.

Code:



```python
# (3) Estimate the Information Gain of all the attributes.
print('\n(3) Estimate the Information Gain of all the attributes:\n')
print('Gain(tennis, Outlook) = ', calc_info_gain(tennis, 'Outlook', 'PlayTennis'))
print('Gain(tennis, Temperature) = ', calc_info_gain(tennis, 'Temperature', 'PlayTennis'))
print('Gain(tennis, Humidity) = ', calc_info_gain(tennis, 'Humidity', 'PlayTennis'))
print('Gain(tennis, Wind) = ', calc_info_gain(tennis, 'Wind', 'PlayTennis'))

print('\nBecause the attribute Outlook has the highest information gain, '
      'so we choose the Outlook as the root node.')
```

Output:



```
(3) Estimate the Information Gain of all the attributes:

Gain(tennis, Outlook) = 0.24674981977443933
Gain(tennis, Temperature) = 0.029222565658955487
Gain(tennis, Humidity) = 0.15183550136234164
Gain(tennis, Wind) = 0.04812703040826544

Because the attribute Outlook has the highest information gain, so we choose the Outlook as the root node.
```

Because the attribute Outlook has the highest information gain, so we choose the Outlook as the root node.

Task 3 Decision Tree Algorithm

Implement the ID3 Decision Tree from the pseudocode (recursive algorithm) below and induce/learn the tree from the data in the Figure.

In this task, I implement the ID3 algorithm and use titanic_passenger_survival.csv to test it. It will output every node in the decision tree.

The output node form:

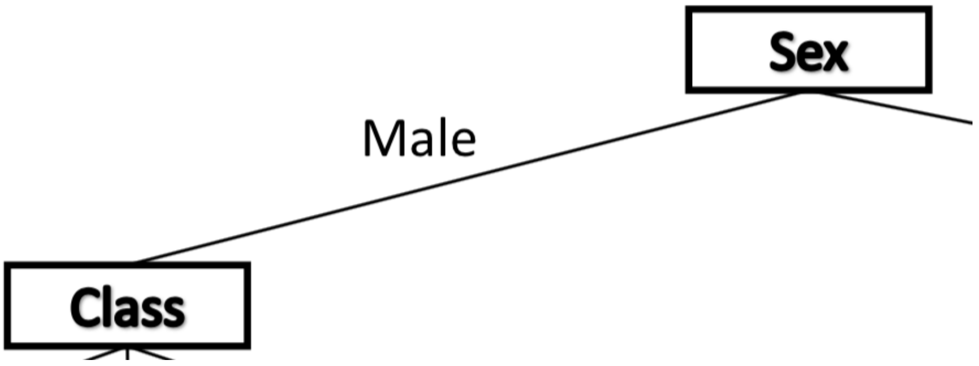
Node <number>: <node value>
Label: <label value>
Parent: <parent value>

For example:

Node 2 : Class
Label: Male
Parent: Node 1 Sex

It denotes the class node with index 2, the label of the node is Male and its parent is Node 1 Sex. (index is used to differentiate the different nodes with the same attribute).

So the paragraph of this node:



The code and the output are in later pages.

task3

November 3, 2018

```
In [4]: import numpy as np
        import pandas as pd

        # Node counter
        NODE_NUMBER = 0

        class Node:
            """
            The class of node in tree, it has three instance attributes.
            is_leaf: denotes if the node is a leaf node.
            node_value: the value of the node.
            sub_node_list: contains key-value list for the sub-node of current node.
                           key denotes the label of the sub-node, value denotes the address of

            """
            def __init__(self):
                global NODE_NUMBER
                NODE_NUMBER = NODE_NUMBER + 1
                self.is_leaf = False
                self.node_value = ''
                self.sub_node_list = {}
                self.id = NODE_NUMBER


        def cal_feature(df, target_label):
            """ Calculate the feature list.

            :param df: the data frame to handle
            :param target_label: the index of target
            :return feature_list: the list of features
            """
            feature_list = list(df.columns)
            feature_list.remove(target_label)

            return feature_list


        def calc_ent(df, target_label):
```

```
    """ Calculate the entropy of df.

    :param df: the data frame to handle
    :param target_label: the index of target
    :return ent: the entropy of s
    """
    df_value_list = set()

    for index, row in df.iterrows():
        df_value_list.add(row[target_label])

    ent = 0.0
    for df_value in df_value_list:
        p = float(df[df.loc[:, target_label] == df_value].shape[0])/df.shape[0]
        log_p = np.log2(p)
        ent -= p * log_p

    return ent


# Calculate the information gain Gain(df, a, target_label).
def calc_info_gain(df, a, target_label):
    """ Calculate the information gain Gain(df, a, target_label).

    :param df: the data frame to calculate
    :param a: the attribute
    :param target_label: the index name of target
    :return: the information gain of df by the given attribute a
    """

    # entropy of df
    ent_df = calc_ent(df, target_label)

    # a_value_list
    a_value_list = set()
    for index, row in df.iterrows():
        a_value_list.add(row[a])

    # Calculate the information gain of attribute a.
    info_gain = ent_df
    for a_value in a_value_list:
        df_a_value = df[df.loc[:,a] == a_value]
        ent_df_a_value = calc_ent(df_a_value, target_label)
        info_gain -= float(df_a_value.shape[0]) / df.shape[0] * ent_df_a_value

    return info_gain
```

```python
def choose_best_attribute(df, features, target_label):
    """ Choose the best attribute according to the information gain.

    :param df: the data frame to handle
    :param features: feature list
    :param target_label: the index of target
    :return best_feature: the best attribute chosen
    """
    best_info_gain = 0
    best_feature = ''

    for attribute in features:
        info_gain = calc_info_gain(df, attribute, target_label)
        if info_gain > best_info_gain:
            best_info_gain = info_gain
            best_feature = attribute

    return best_feature


def split_data(df, feature, feature_value):
    """ Split the data by the feature value.

    :param df: the data frame to handle
    :param feature: the feature chosen to split the data
    :param feature_value: the value of the feature
    :return: the data with the specified feature value
    """
    new_df = df[df.loc[:, feature] == feature_value]
    return new_df


# Record all the possible value of all features.
def calculate_possible_value(df, attributes):
    """ Record all the possible value of all features.

    :param df: the data frame to handle
    :param attributes: attribute list
    :return: a dictionary with the features and its possible values
    """
    attribute_value_dictionary = {}
    for attribute in attributes:
        possible_value = df[attribute].drop_duplicates()
        attribute_value_dictionary[attribute] = list(possible_value)
    return attribute_value_dictionary


def test_same_value(df, target_label):
    """ Test if the observations have the same targets.

    :param df: the data frame to handle
    :param target_label: the index of target
    :return: the target value without duplication
    """
    data = df[target_label].drop_duplicates()
    return data


def create_tree(df, target_label, attributes):
    """ Generate a decesion tree given by df, target_label, attributes.

    :param df: the data frame to handle
    :param target_label: the index of target
    :param attributes: attribute list
    :return: a node
    """
    # Create a node.
    node = Node()

    # Get the target value without duplication.
    df_target = test_same_value(df, target_label)

    if len(df_target) == 1:
        # case 1: the observations have the same target value.
        node.node_value = df_target.values[0]
        node.is_leaf = True
    elif len(attributes) == 0:
        # case 2: there are no remained attributes.
        node.node_value = df[target_label].mode().iloc[0]
        node.is_leaf = True
    else:
        # other case: choose the best attribute as node
        best_attribute = choose_best_attribute(df, attributes, target_label)
        node.node_value = best_attribute
        node.is_leaf = False
        best_attribute_value_list = dictionary[best_attribute]
        attributes.remove(best_attribute)

        # For each attribute value, generate the sub branch.
        for attribute_value in best_attribute_value_list:
            # Split the data by the attribute value.
            sub_df = split_data(df, best_attribute, attribute_value)
            if len(sub_df) == 0:
                # Sub-branch has no observations.
                sub_node = Node()
                sub_node.is_leaf = True
```

```
                    sub_node.node_value = df[target_label].mode().iloc[0]
                    node.sub_node_list[attribute_value] = sub_node
                else:
                    # Sub-branch has observations.
                    new_attributes = attributes.copy()
                    node.sub_node_list[attribute_value] = create_tree(sub_df, target_label
        return node


def print_tree(node, label=None, parent=None):
    """ Print the information of the tree.

    :param node: the node to print
    :param label: the label of the node
    :param parent: the parent of the node
    :return:
    """
    print('Node', node.id, ': ', node.node_value)
    if (label is not None) and (parent is not None):
        print('Label: ', label, '\nParent: Node', parent.id, parent.node_value, '\n')
    else:
        print('Label: None \nParent: None\n')

    if len(node.sub_node_list) == 0:
        return
    else:
        for label, sub_node in node.sub_node_list.items():
            print_tree(sub_node, label, node)


# Use the titanic_passenger_survival data.
df = pd.read_csv('titanic_passenger_survival.csv', header=0, usecols=[0, 1, 2, 3])
target_label = 'Survived'

# Use the tennis data.
# df = pd.read_csv('tennis.csv', header=0, usecols=[1,2,3,4,5])
# target_label = 'PlayTennis'

features = cal_feature(df, target_label)    # Obtain the feature list
dictionary = calculate_possible_value(df, features)     # Get all the possible value o

# Generate the tree using ID3 algorithm.
tree = create_tree(df, target_label, features)

# Print the tree.
print_tree(tree)
```

```
Node 1 :  Sex
```

```
Label: None
Parent: None

Node 2 :  Class
Label:  Male
Parent: Node 1 Sex

Node 3 :  Age
Label:  3rd
Parent: Node 2 Class

Node 4 :  No
Label:  Child
Parent: Node 3 Age

Node 5 :  No
Label:  Adult
Parent: Node 3 Age

Node 6 :  Age
Label:  1st
Parent: Node 2 Class

Node 7 :  Yes
Label:  Child
Parent: Node 6 Age

Node 8 :  No
Label:  Adult
Parent: Node 6 Age

Node 9 :  Age
Label:  2nd
Parent: Node 2 Class

Node 10 :  Yes
Label:  Child
Parent: Node 9 Age

Node 11 :  No
Label:  Adult
Parent: Node 9 Age

Node 12 :  Class
Label:  Female
Parent: Node 1 Sex

Node 13 :  Age
```

```
Label:  3rd
Parent: Node 12 Class

Node 14 :  No
Label:  Child
Parent: Node 13 Age

Node 15 :  No
Label:  Adult
Parent: Node 13 Age

Node 16 :  Age
Label:  1st
Parent: Node 12 Class

Node 17 :  Yes
Label:  Child
Parent: Node 16 Age

Node 18 :  Yes
Label:  Adult
Parent: Node 16 Age

Node 19 :  Age
Label:  2nd
Parent: Node 12 Class

Node 20 :  Yes
Label:  Child
Parent: Node 19 Age

Node 21 :  Yes
Label:  Adult
Parent: Node 19 Age
```