Pokok Bahasan IV
Decision Tree with Python

Kode Pokok Bahasan: TIK.RPL03.001.003.01

Deskripsi Pokok Bahasan:
Membahas bagaimana pembuatan Decision Tree pada dataset yang diberikan.

No	Elemen Kompetensi	Indikator Kinerja	Jml Jam	Hal
1	Menerapkan Decision Tree pada data Weather Nominal dataset 1.	1.1 Membuat Decision Tree menggunakan Library Scikit-Learn menggunakan data Weather Nominal	1	12
2	Menerapkan prediksi dan class pada Decision Tree data Weather Nominal dataset 2.	1.1 Membuat Decision Tree menggunakan Library Scikit-Learn menggunakan data Weather Nominal 1.2 Membuat prediksi dari Decision Tree 1.3 Membuat class dari prediksi	2	15

TUGAS PENDAHULUAN
Hal yang harus dilakukan dan acuan yang harus dibaca sebelum praktikum :
Menginstal Anaconda Python pada PC masing-masing praktikan.
2. Menginstal Jupyter Notebook pada Anaconda Python pada PC masing-masing praktikan.

DAFTAR PERTANYAAN
1. Apa itu decision tree?
2. Apa kegunaan Decision tree?
3. Adakah perbedaan dalam membuat decision tree antara R dan Python, berikan alasannya!

1. Decision tree merupakan suatu struktur yang digunakan untuk membantu proses pengambilan keputusan. Disebut sebagai "tree" karena struktur ini menyerupai sebuah pohon lengkap dengan akar, batang, dan percabangannya.

2. Manfaat utama dari penggunaan decision tree adalah kemampuannya untuk mem-break down proses pengambilan keputusan yang kompleks menjadi lebih simple, sehingga pengambil keputusan akan lebih menginterpretasikan solusi dari permasalahan.

3. Perbedaan yang sangat terlihat dari kedua bahasa ini adalah R digunakan untuk analisis statistik, sedangkan Python lebih general. Python adalah bahasa multipurpose, sama seperti C++ dan Java. Penggunaannya pun cenderung lebih mudah dipelajari, tak seperti R yang lebih kompleks.

TEORI SINGKAT

Decision tree adalah salah satu metode klasifikasi yang paling populer, karena mudah untuk diinterpretasi oleh manusia. Decision tree adalah model prediksi menggunakan struktur pohon atau struktur berhirarki.

Konsep dari pohon keputusan adalah mengubah data menjadi decision tree dan aturan-aturan keputusan. Manfaat utama dari penggunaan decision tree adalah kemampuannya untuk mem-break down proses pengambilan keputusan yang kompleks menjadi lebih simple, sehingga pengambil keputusan akan lebih menginterpretasikan solusi dari permasalahan.

Salah satu algoritma dari decision tree adalah CART (Classification and Regression Tree). Dimana metode ini merupakan gabungan dari dua jenis pohon, yaitu classification tree dan juga regression tree. Untuk memudahkan, berikut ilustrasi dari keduanya.

LAB SETUP

Hal yang harus disiapkan dan dilakukan oleh praktikan untuk menjalankan praktikum modul ini.

1. Menginstall library yang dibutuhkan untuk mengerjakan modul.
2. Menjalankan R Studio.

ELEMEN KOMPETENSI I

Deskripsi:
Menerapkan Decision Tree pada data Weather Nominal dataset 1.

Kompetensi Dasar:
Membuat Decision Tree menggunakan Library Scikit-Learn menggunakan data Weather Nominal.

Latihan 1.1.1

Penjelasan Singkat :
Pada latihan ini anda akan diminta untuk membangun decision tree menggunakan library yang disediakan oleh Python.

Langkah-Langkah Praktikum:

1. Buat file baru pada Jupyter Notebook atau Google Colab https://colab.research.google.com/

2. Letakkan file data yang digunakan pada direktori yang sama. Jika menggunakan Google Colab, upload data terlebih dahulu session storage.
   Klik upload to session storage seperti pada gambar di bawah.

   Maka akan muncul seperti di bawah ini, pilihlah data prak4.csv pada penyimpanan anda, kemudian klik open.

   Jika sudah makan akan muncul pada side bar seperti gambar berikut.

3. Import library python yang akan digunakan.

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier #import decision tree classifier
from sklearn.model_selection import train_test_split #import train_test_split function
from sklearn import metrics
```

```
In [1]: import pandas as pd
        from sklearn.tree import DecisionTreeClassifier #import decision tree classifier
        from sklearn.model_selection import train_test_split #import train_test_split function
        from sklearn import metrics
```

4. Membaca data prak4.csv dengan metode read_csv

```
data_namapraktikan = pd.read_csv("prak4.csv")
data_namapraktikan.head()
```

```
In [5]: data_ibnu = pd.read_csv("D:/File Kuliah Semester 5/Penambangan Data/Prak-4/prak4.csv")
        data_ibnu.head()
```

Out[5]:

	cuaca	suhu	kelembaban	berangin	bermain
0	cerah	panas	Tinggi	salah	tidak
1	cerah	panas	Tinggi	benar	tidak
2	berawan	panas	Tinggi	salah	ya
3	hujan	sejuk	Tinggi	salah	ya
4	hujan	dingin	Normal	salah	ya

5. Mengecek data secara umum

```
data_namapraktikan.info()
```

```
In [6]: data_.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14 entries, 0 to 13
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   cuaca       14 non-null     object
 1   suhu        14 non-null     object
 2   kelembaban  14 non-null     object
 3   berangin    14 non-null     object
 4   bermain     14 non-null     object
dtypes: object(5)
memory usage: 688.0+ bytes
```

6. Pada saat harus bekerja dengan kolom atau atribut bertipe kolom, maka data harus diubah dalam format numerik. Pada library sklearn.preprocessing tersedia dua metode untuk mengubah tipe data kategorikal menjadi data numerik yaitu label encoding dan one hot encoding.
   6.1. Import library sklearn untuk melakukan label encoding dan onehot encoding. Kemudian melakukan datapreprocessing menggunakan label encoding dan one hot encoding.

```
from numpy import array
from numpy import argmax
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing  import OneHotEncoder

print(data_namapraktikan.columns)
fiturs = ['cuaca', 'suhu', 'kelembaban', 'berangin']
labels_encoder = LabelEncoder()
jobs_encoder = OneHotEncoder(sparse=False)
data_namapraktikan1 = data_namapraktikan.copy(deep=True)
for fitur in fiturs:
  print(f'{fitur}:{data_namapraktikan1[fitur].unique()}')
  values = array(data_namapraktikan1[fitur])
  integer_encoded = labels_encoder.fit_transform(values)
  transformed = jobs_encoder.fit_transform(integer_encoded.reshape(len(i
    nteger_encoded), 1))
  ohe_df = pd.DataFrame(transformed,columns=jobs_encoder.get_feature
    _names())
  data_namapraktikan1 = pd.concat([data_namapraktikan1, ohe_df], axis=1
    ).drop([fitur], axis=1)

data_namapraktikan1.head()
```

In [20]:
```
print(data_ibnu.columns)
fiturs = ['cuaca', 'suhu', 'kelembaban', 'berangin']
labels_encoder = LabelEncoder()
jobs_encoder = OneHotEncoder(sparse=False)
data_ibnu1 = data_ibnu.copy(deep=True)
for fitur in fiturs:
    print(f'{fitur}:{data_ibnu1[fitur].unique()}')
    values = array(data_ibnu1[fitur])
    integer_encoded = labels_encoder.fit_transform(values)
    transformed = jobs_encoder.fit_transform(integer_encoded.reshape(len(integer_encoded), 1))
    ohe_df = pd.DataFrame(transformed,columns=jobs_encoder.get_feature_names())
    data_ibnu1 = pd.concat([data_ibnu1, ohe_df], axis=1).drop([fitur], axis=1)

data_ibnu1.head()
```

```
Index(['cuaca', 'suhu', 'kelembaban', 'berangin', 'bermain'], dtype='object')
cuaca:['cerah' 'berawan' 'hujan']
suhu:['panas' 'sejuk' 'dingin']
kelembaban:['Tinggi' 'Normal']
berangin:['salah' 'benar']
```

Out[20]:

	bermain	x0_0	x0_1	x0_2	x0_0	x0_1	x0_2	x0_0	x0_1	x0_0	x0_1
0	tidak	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0
1	tidak	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0
2	ya	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0
3	ya	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0
4	ya	0.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0

6.2. Mengubah label pada kolom bermain dengan label encoder.

```
integer_encoded = labels_encoder.fit_transform(data_namapraktikan1['bermain'])
main = pd.DataFrame(integer_encoded,columns=['label_bermain',])
data_namapraktikan1 = pd.concat([data_namapraktikan1, main], axis=1).drop(['bermain'], axis=1)
data_namapraktikan1.head()
```

In [9]:
```
integer_encoded = labels_encoder.fit_transform(data_ibnu1['bermain'])
main = pd.DataFrame(integer_encoded,columns=['label_bermain',])
data_ibnu1 = pd.concat([data_ibnu1, main], axis=1).drop(['bermain'], axis=1)
data_ibnu1.head()
```

Out[9]:

	x0_0	x0_1	x0_2	x0_0	x0_1	x0_2	x0_0	x0_1	x0_0	x0_1	label_bermain
0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0
1	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	0
2	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	1
3	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	1
4	0.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	1

6.3. Menyiapkan fitur untuk prediksi dan label klasifikasi.

Data untuk prediksi yaitu data pada dataset kecuali kolom “bermain”. Data untuk label klasifikasi yaitu data pada kolom “bermain”.

```
y = data_namapraktikan1['label_bermain']
x = data_namapraktikan1.drop(columns=['label_bermain'])
```

```
print(f'variabel prediktor:{x.columns}')
print(f'label klasifikasi:{y.name}')
```

```
In [10]: y = data_ibnu1['label_bermain']
         x = data_ibnu1.drop(columns=['label_bermain'])

         print(f'variabel prediktor:{x.columns}')
         print(f'label klasifikasi:{y.name}')

         variabel prediktor:Index(['x0_0', 'x0_1', 'x0_2', 'x0_0', 'x0_1', 'x0_2', 'x0_0', 'x0_1', 'x0_0',
                'x0_1'],
               dtype='object')
         label klasifikasi:label_bermain
```

7. Membagi dataset menjadi data training dan data testing.
   Data training = X_train dan y_train
   Data testing = x_test dan y_test
   Data training : data testing = 70 : 30 sehingga test_size = 0.3

```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_stat
e=1)
size_xtrain = len(X_train)
size_xtest = len(X_test)
print(f'ukuran data training: {size_xtrain} ({size_xtrain/(size_xtrain+size_xtest)})')
print(f'ukuran data testing: {size_xtest} ({size_xtest/(size_xtrain+size_xtest)})')
```

```
In [11]: X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=1)
         size_xtrain = len(X_train)
         size_xtest = len(X_test)
         print(f'ukuran data training: {size_xtrain} ({size_xtrain/(size_xtrain+size_xtest)})')
         print(f'ukuran data testing: {size_xtest} ({size_xtest/(size_xtrain+size_xtest)})')

         ukuran data training: 9 (0.6428571428571429)
         ukuran data testing: 5 (0.35714285714285715)
```

8. Melakukan split training dan testing model dengan membuat klasifier.

```
clf_namapraktikan = DecisionTreeClassifier()

# proses training model
clf_namapraktikan = clf_namapraktikan.fit(X_train,y_train)

#proses testing model
y_pred = clf_namapraktikan.predict(X_test)
```

```
In [12]: clf_ibnu = DecisionTreeClassifier()

         # proses training model
         clf_ibnu = clf_ibnu.fit(X_train,y_train)

         #proses testing model
         y_pred = clf_ibnu.predict(X_test)
```

9. Membuat confusion matrix.

```
from sklearn.metrics import confusion_matrix

tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
print(f'confusion matrix')

print(f'{tp} | {fp}')
print(f'{fn} | {tn}')
```

```
In [13]: from sklearn.metrics import confusion_matrix

         tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
         print(f'confusion matrix')

         print(f'{tp} | {fp}')
         print(f'{fn} | {tn}')

confusion matrix
3 | 0
1 | 1
```

10. Melihat keakuratan model

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
In [14]: print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

Accuracy: 0.8
```

11. Menampilkan decision tree

```
import numpy as np
from matplotlib import pyplot as plt
from sklearn import tree

n_nodes = clf_namapraktikan.tree_.node_count
children_left = clf_namapraktikan.tree_.children_left
children_right = clf_namapraktikan.tree_.children_right
feature = clf_namapraktikan.tree_.feature
```

```
threshold = clf_namapraktikan.tree_.threshold

node_depth = np.zeros(shape=n_nodes, dtype=np.int64)
is_leaves = np.zeros(shape=n_nodes, dtype=bool)
stack = [(0, 0)]  # start with the root node id (0) and its depth (0)
while len(stack) > 0:
  # `pop` ensures each node is only visited once
  node_id, depth = stack.pop()
  node_depth[node_id] = depth

  # If the left and right child of a node is not the same we have a split
  # node
  is_split_node = children_left[node_id] != children_right[node_id]    # If a split
node, append left and right children and depth to `stack`
  # so we can loop through them
  if is_split_node:
    stack.append((children_left[node_id], depth + 1))
    stack.append((children_right[node_id], depth + 1))
  else:
    is_leaves[node_id] = True

print("The binary tree structure has {n} nodes and has "
    "the following tree structure:\n".format(n=n_nodes))
for i in range(n_nodes):
  if is_leaves[i]:
    print("{space}node={node} is a leaf node.".format(
      space=node_depth[i] * "\t", node=i))
  else:
    print("{space}node={node} is a split node: "
        "go to node {left} if X[:, {feature}] <= {threshold} "
        "else to node {right}.".format(
          space=node_depth[i] * "\t",
          node=i,
          left=children_left[i],
          feature=feature[i],
          threshold=threshold[i],
          right=children_right[i]))
```

12. Menampilkan plot decision tree

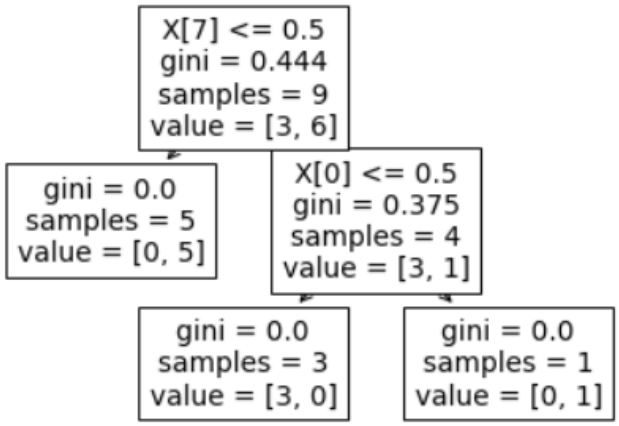
```
tree.plot_tree(clf_namapraktikan)
plt.show()
```

In [16]:
```
n_nodes = clf_ibnu.tree_.node_count
children_left = clf_ibnu.tree_.children_left
children_right = clf_ibnu.tree_.children_right
feature = clf_ibnu.tree_.feature
threshold = clf_ibnu.tree_.threshold

node_depth = np.zeros(shape=n_nodes, dtype=np.int64)
is_leaves = np.zeros(shape=n_nodes, dtype=bool)
stack = [(0, 0)]  # start with the root node id (0) and its depth (0)
while len(stack) > 0:
    # `pop` ensures each node is only visited once
    node_id, depth = stack.pop()
    node_depth[node_id] = depth

    # If the left and right child of a node is not the same we have a split
    # node
    is_split_node = children_left[node_id] != children_right[node_id]    # If a split node, append left and right children and de
    # so we can loop through them
    if is_split_node:
        stack.append((children_left[node_id], depth + 1))
        stack.append((children_right[node_id], depth + 1))
    else:
        is_leaves[node_id] = True

print("The binary tree structure has {n} nodes and has "
      "the following tree structure:\n".format(n=n_nodes))
for i in range(n_nodes):
    if is_leaves[i]:
        print("{space}node={node} is a leaf node.".format(
            space=node_depth[i] * "\t", node=i))
    else:
        print("{space}node={node} is a split node: "
              "go to node {left} if X[:, {feature}] <= {threshold} "
              "else to node {right}.".format(
                  space=node_depth[i] * "\t",
                  node=i,
```

```
            space=node_depth[i] * "\t", node=i))
    else:
        print("{space}node={node} is a split node: "
              "go to node {left} if X[:, {feature}] <= {threshold} "
              "else to node {right}.".format(
                  space=node_depth[i] * "\t",
                  node=i,
                  left=children_left[i],
                  feature=feature[i],
                  threshold=threshold[i],
                  right=children_right[i]))
```

```
The binary tree structure has 5 nodes and has the following tree structure:

node=0 is a split node: go to node 1 if X[:, 7] <= 0.5 else to node 2.
        node=1 is a leaf node.
        node=2 is a split node: go to node 3 if X[:, 0] <= 0.5 else to node 4.
                node=3 is a leaf node.
                node=4 is a leaf node.
```

```
In [17]: tree.plot_tree(clf_ibnu)
         plt.show()
```



13. Melakukan klasifikasi berdasarkan entropi

```
clf1 = DecisionTreeClassifier(criterion="entropy")

# Train Decision Tree Classifer
clf1.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf1.predict(X_test)

tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
print(f'confusion matrix')
print(f'{tp} | {fp}')
print(f'{fn} | {tn}')

#menampilkan akurasi model
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
In [18]: clf1 = DecisionTreeClassifier(criterion="entropy")

         # Train Decision Tree Classifer
         clf1.fit(X_train,y_train)

         #Predict the response for test dataset
         y_pred = clf1.predict(X_test)

         tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
         print(f'confusion matrix')
         print(f'{tp} | {fp}')
         print(f'{fn} | {tn}')

         #menampilkan akurasi model
         print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

         confusion matrix
         3 | 0
         1 | 1
         Accuracy: 0.8
```

Interpretasi output decision tree :

Interpretasi nya adalah jika nilai x lebih dari 0 maka akan dipisahkan, seterusnya akan begitu sampai nilai x tidak lebih dari 0, nilai gini index juga ketika sudah mulai dibagi, maka nilainya akan terus berkurang sampai terbagi rata.

ELEMEN KOMPETENSI II

Deskripsi:
Menerapkan prediksi menggunakan Decision Tree pada data Weather Nominal dataset 2.

Kompetensi Dasar:
Membuat prediksi pada Tree menggunakan data Weather Nominal dataset 2.

Latihan 1.2.1

Penjelasan Singkat :
Pada latihan ini anda akan diminta untuk membangun decision tree dan melakukan prediksi menggunakan library yang disediakan oleh python.

Langkah-Langkah Praktikum:

1. Disediakan data sebagai berikut :
Link dataset : https://www.kaggle.com/datasets/altruistdelhite04/loan-prediction-problem-dataset

2. Buatlah decision tree untuk dataset di atas menggunakan langkah-langkah pada elemen kompetensi 1. Tampilkan langkah-langkah pembuatannya beserta hasil decision tree.

In [21]:
```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier #import decision tree classifier
from sklearn.model_selection import train_test_split #import train_test_split function
from sklearn import metrics
```

In [16]:
```
data_ibnu = pd.read_csv("D:/File Kuliah Semester 5/Penambangan Data/Prak-4/trainers.csv")
data_ibnu.head()
```

Out[16]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Ar
0	Male	No	0	Graduate	No	5849	0.0	480	360	1	Urt
1	Male	Yes	1	Graduate	No	4583	1508.0	128	360	1	Ru
2	Male	Yes	0	Graduate	Yes	3000	0.0	66	360	1	Urt
3	Male	Yes	0	Not Graduate	No	2583	2358.0	120	360	1	Urt
4	Male	No	0	Graduate	No	6000	0.0	141	360	1	Urt

In [17]:
```
data_ibnu.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Gender             614 non-null    object
 1   Married            614 non-null    object
 2   Dependents         614 non-null    int64
 3   Education          614 non-null    object
 4   Self_Employed      614 non-null    object
 5   ApplicantIncome    614 non-null    int64
 6   CoapplicantIncome  614 non-null    float64
 7   LoanAmount         614 non-null    int64
 8   Loan_Amount_Term   614 non-null    int64
 9   Credit_History     614 non-null    int64
 10  Property_Area      614 non-null    object
 11  Loan_Status        614 non-null    object
dtypes: float64(1), int64(5), object(6)
memory usage: 57.7+ KB
```

In [22]:
```
from numpy import array
from numpy import argmax
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing  import OneHotEncoder
```

In [18]:
```
print(data_ibnu.columns)
```

```
Index(['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
       'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

In [28]:
```
print(data_ibnu.columns)
fiturs = ['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
       'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area']
labels_encoder = LabelEncoder()
jobs_encoder = OneHotEncoder(sparse=False)
data_ibnu1 = data_ibnu.copy(deep=True)
for fitur in fiturs:
    print(f'{fitur}:{data_ibnu1[fitur].unique()}')
    values = array(data_ibnu1[fitur])
    integer_encoded = labels_encoder.fit_transform(values)
    transformed = jobs_encoder.fit_transform(integer_encoded.reshape(len(integer_encoded), 1))
    ohe_df = pd.DataFrame(transformed,columns=jobs_encoder.get_feature_names())
    data_ibnu1 = pd.concat([data_ibnu1, ohe_df], axis=1).drop([fitur], axis=1)

data_ibnu1.head()
```

```
 300 376 117  71 173  46 228 308 236 570 380 296 156 103  45  65  53  62
 218 178 239 405 148 190 149 153 162 230  86 234 246 500 186 119 107 209
 208 243  40 250 311 400 161 196 324 157 145 181  26 211   9 205  36  61
 146 292 142 350 253]
Loan_Amount_Term:[360 120 240 480 100 180  60 300  36  84  12]
Credit_History:[1 0]
Property_Area:['Urban' 'Rural' 'Semiurban']
```

Out[28]:

	Loan_Status	x0_0	x0_1	x0_0	x0_1	x0_0	x0_1	x0_2	x0_3	x0_0	...	x0_6	x0_7	x0_8	x0_9	x0_10	x0_0	x0_1	x0_0	x0_1	x0_2
0	Y	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	...	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0
1	N	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	...	0.0	0.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0
2	Y	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	...	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0
3	Y	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0
4	Y	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	...	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0

In [29]:
```
integer_encoded = labels_encoder.fit_transform(data_ibnu1['Loan_Status'])
main = pd.DataFrame(integer_encoded,columns=['label_loan_status',])
data_ibnu1 = pd.concat([data_ibnu1, main], axis=1).drop(['Loan_Status'], axis=1)
data_ibnu1.head()
```

Out[29]:

	x0_0	x0_1	x0_0	x0_1	x0_0	x0_1	x0_2	x0_3	x0_0	x0_1	...	x0_7	x0_8	x0_9	x0_10	x0_0	x0_1	x0_0	x0_1	x0_2	label_loan_status
0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	1
1	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0
2	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	1
3	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	1
4	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	1

In [32]:
```
y = data_ibnu1['label_loan_status']
x = data_ibnu1.drop(columns=['label_loan_status'])

print(f'variabel prediktor:{x.columns}')
print(f'label klasifikasi:{y.name}')
```

```
variabel prediktor:Index(['x0_0', 'x0_1', 'x0_0', 'x0_1', 'x0_0', 'x0_1', 'x0_2', 'x0_3', 'x0_0',
       'x0_1',
       ...
       'x0_6', 'x0_7', 'x0_8', 'x0_9', 'x0_10', 'x0_0', 'x0_1', 'x0_0', 'x0_1',
       'x0_2'],
      dtype='object', length=1023)
label klasifikasi:label_loan_status
```

In [33]:
```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=1)
size_xtrain = len(X_train)
size_xtest = len(X_test)
print(f'ukuran data training: {size_xtrain} ({size_xtrain/(size_xtrain+size_xtest)})')
print(f'ukuran data testing: {size_xtest} ({size_xtest/(size_xtrain+size_xtest)})')
```

```
ukuran data training: 429 (0.6986970684039088)
ukuran data testing: 185 (0.30130293159609123)
```

In [34]:
```
clf_ibnu = DecisionTreeClassifier()

# proses training model
clf_ibnu = clf_ibnu.fit(X_train,y_train)

#proses testing model
y_pred = clf_ibnu.predict(X_test)
```

In [35]:
```
from sklearn.metrics import confusion_matrix

tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
print(f'confusion matrix')

print(f'{tp} | {fp}')
print(f'{fn} | {tn}')
```

```
confusion matrix
115 | 33
9 | 28
```

In [36]:
```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

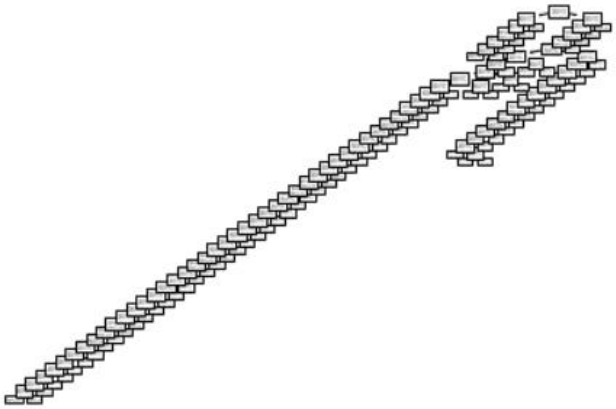
```
Accuracy: 0.772972972972973
```

In [37]:
```
import numpy as np
from matplotlib import pyplot as plt
from sklearn import tree
```

In [38]:
```
n_nodes = clf_ibnu.tree_.node_count
children_left = clf_ibnu.tree_.children_left
children_right = clf_ibnu.tree_.children_right
feature = clf_ibnu.tree_.feature
threshold = clf_ibnu.tree_.threshold

node_depth = np.zeros(shape=n_nodes, dtype=np.int64)
is_leaves = np.zeros(shape=n_nodes, dtype=bool)
stack = [(0, 0)]  # start with the root node id (0) and its depth (0)
while len(stack) > 0:
    # `pop` ensures each node is only visited once
    node_id, depth = stack.pop()
    node_depth[node_id] = depth

    # If the left and right child of a node is not the same we have a split
    # node
    is_split_node = children_left[node_id] != children_right[node_id]    # If a split node, append left and right children and de
    # so we can loop through them
    if is_split_node:
        stack.append((children_left[node_id], depth + 1))
        stack.append((children_right[node_id], depth + 1))
    else:
        is_leaves[node_id] = True

print("The binary tree structure has {n} nodes and has "
      "the following tree structure:\n".format(n=n_nodes))
for i in range(n_nodes):
    if is_leaves[i]:
        print("{space}node={node} is a leaf node.".format(
            space=node_depth[i] * "\t", node=i))
    else:
        print("{space}node={node} is a split node: "
              "go to node {left} if X[:, {feature}] <= {threshold} "
              "else to node {right}.".format(
                  space=node_depth[i] * "\t",
                  node=i,
```

```
The binary tree structure has 149 nodes and has the following tree structure:

node=0 is a split node: go to node 1 if X[:, 1019] <= 0.5 else to node 12.
        node=1 is a split node: go to node 2 if X[:, 570] <= 0.5 else to node 11.
                node=2 is a split node: go to node 3 if X[:, 58] <= 0.5 else to node 10.
                        node=3 is a split node: go to node 4 if X[:, 345] <= 0.5 else to node 9.
                                node=4 is a split node: go to node 5 if X[:, 305] <= 0.5 else to node 8.
                                        node=5 is a split node: go to node 6 if X[:, 513] <= 0.5 else to node 7.
                                                node=6 is a leaf node.
                                                node=7 is a leaf node.
                                        node=8 is a leaf node.
                                node=9 is a leaf node.
                        node=10 is a leaf node.
                node=11 is a leaf node.
        node=12 is a split node: go to node 13 if X[:, 318] <= 0.5 else to node 148.
                node=13 is a split node: go to node 14 if X[:, 103] <= 0.5 else to node 147.
                        node=14 is a split node: go to node 15 if X[:, 712] <= 0.5 else to node 146.
                                node=15 is a split node: go to node 16 if X[:, 466] <= 0.5 else to node 145.
                                        node=16 is a split node: go to node 17 if X[:, 1021] <= 0.5 else to node 116.
```

In [39]:
```
tree.plot_tree(clf_ibnu)
plt.show()
```



In [40]:
```
clf1 = DecisionTreeClassifier(criterion="entropy")

# Train Decision Tree Classifer
clf1.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf1.predict(X_test)

tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
print(f'confusion matrix')
print(f'{tp} | {fp}')
print(f'{fn} | {tn}')

#menampilkan akurasi model
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
confusion matrix
108 | 34
16 | 27
Accuracy: 0.7297297297297297
```

3. Berikan penjelasan tentang output yang muncul!

Interpretasi output decision tree :

Interpretasi nya adalah jika nilai x lebih dari 0 maka akan dipisahkan, seterusnya akan begitu sampai nilai x tidak lebih dari 0, nilai gini index juga ketika sudah mulai dibagi, maka nilainya akan terus berkurang sampai terbagi rata.

CEK LIST

Elemen Kompetensi	No Latihan	Penyelesaian	
		Selesai	Tidak selesai
1	1.1.1	✓	
2	1.2.1	✓	

FORM UMPAN BALIK

Elemen Kompetensi	**Tingkat Kesulitan**	**Tingkat Ketertarikan**	**Waktu Penyelesaian dalam menit**
Memahami decision tree	☐ Sangat Mudah ☐ Mudah ☐ ✓ Biasa ☐ Sulit ☐ Sangat Sulit	☐ Tidak Tertarik ☐ Cukup Tertarik ☐ Tertarik ☐ ✓ Sangat Tertarik	
Mengimplementasikan decision tree.	☐ Sangat Mudah ☐ Mudah ☐ ✓ Biasa ☐ Sulit ☐ Sangat Sulit	☐ Tidak Tertarik ☐ Cukup Tertarik ☐ Tertarik ☐ ✓ Sangat Tertarik	