**AFL 3 *Data Mining***

**Pengembangan Mandiri *Tasks Data Mining***

****

Oleh :

Steven Ongkowidjojo – 0706022110035

***Information Systems for Business***

**Universitas Ciputra Surabaya**

**2024**

**Daftar Isi**

**BAB I *Decision Tree*1**

1. *Boosting*2
2. Tujuan *Boosting*3
3. Kekhususan *Boosting*3
4. Keuntungan *Boosting*3
5. Prinsip Kerja *Boosting*3
6. Uji Coba *Boosting*3
7. Hasil *Boosting*3
8. *Bagging*2
9. Tujuan *Bagging*3
10. Kekhususan *Bagging*3
11. Keuntungan *Bagging*3
12. Prinsip Kerja *Bagging*3
13. Uji Coba *Bagging*3
14. Hasil *Bagging*3
15. *Random Forest*2
16. Tujuan *Random Forest*3
17. Kekhususan *Random Forest*3
18. Keuntungan *Random Forest*3
19. Prinsip Kerja *Random Forest*3
20. Uji Coba *Random Forest*3
21. Hasil *Random Forest*3
22. *Regression Tree*2
23. Tujuan *Regression Tree*3
24. Kekhususan *Regression Tree*3
25. Keuntungan *Regression Tree*3
26. Prinsip Kerja *Regression Tree*3
27. Uji Coba *Regression Tree*3
28. Hasil *Regression Tree*3
29. Kesimpulan2

**BAB II Clustering4**

Type chapter title (level 2)5

Type chapter title (level 3)6

**BAB III Association Rules Mining1**

Type chapter title (level 2)2

Type chapter title (level 3)3

BAB I

*Decision Tree*

1. *Boosting*
2. Tujuan *Boosting*

*Boosting* merupakan sebuah teknik dalam *Data Mining* yang digunakan untuk meningkatkan kinerja model prediksi dengan cara menggabungkan beberapa model lemah menjadi satu model kuat. Tujuan utama dari *Boosting* ini sendiri adalah meningkatkan performa prediktif model dan mengurangi *overfitting*. Dengan memperkuat area yang sering salah diprediksi, teknik *Boosting* dapat menghasilkan model yang lebih akurat dan generalisasi yang baik terhadap data baru.

1. Kekhususan *Boosting*

*Boosting* adalah metode khusus dalam data mining yang bertujuan meningkatkan performa model prediktif dengan menggabungkan serangkaian model lemah secara iteratif. Dengan memberikan bobot lebih pada data yang sulit diprediksi oleh model sebelumnya, *Boosting* mampu mengurangi kesalahan prediksi dan meningkatkan akurasi secara signifikan. Keunikan metode ini terletak pada kemampuannya untuk memfokuskan model pada kesalahan yang telah terjadi sebelumnya, sehingga setiap iterasi menghasilkan model yang semakin kuat dan mampu menangani kompleksitas data dengan lebih baik, menghasilkan prediksi yang lebih akurat dan generalisasi yang lebih baik terhadap dataset yang baru.

1. Keuntungan *Boosting*

Keuntungan utama dari metode *Boosting* dalam data mining adalah kemampuannya untuk meningkatkan akurasi prediksi model. Dengan secara iteratif memfokuskan pada data yang sulit diprediksi, *Boosting* dapat mengurangi kesalahan prediksi dan meningkatkan kinerja model. Hal ini membantu mengatasi masalah *overfitting*, sehingga model dapat lebih baik dalam menangani kompleksitas data dan memberikan hasil yang lebih umum. Selain itu, *Boosting* juga dapat meningkatkan stabilitas model dan mengurangi *variance*, sehingga menghasilkan model yang lebih *robust* dan handal dalam berbagai situasi.

1. Prinsip Kerja *Boosting*

Prinsip kerja boosting adalah menggabungkan beberapa model prediktif lemah menjadi satu model yang lebih kuat. Prosesnya dilakukan secara iteratif, dimulai dengan membuat model lemah, seperti pohon keputusan yang dangkal, yang kemudian diberikan bobot pada data yang salah diprediksi. Model berikutnya akan lebih fokus pada kesalahan yang dilakukan oleh model sebelumnya, dan bobot diberikan kembali. Langkah ini diulang, sehingga setiap model berusaha memperbaiki kesalahan prediksi sebelumnya. Akhirnya, model-model ini digabungkan secara adaptif untuk membentuk model yang lebih kuat dan akurat. Prinsip ini membantu mengatasi kesalahan prediksi, meningkatkan akurasi, dan membuat model lebih mampu menangani kompleksitas data dengan baik.

1. Uji Coba *Boosting*

Dataset yang akan digunakan dalam proyek ini adalah dataset untuk memprediksi performa dari mahasiswa. Untuk melihat lebih lengkap, dataset dapat diakses pada laman website <https://github.com/StevenO29/Tasks-Data-Mining/blob/main/drug200.csv>. Contoh dataset sebagai berikut:



Berikut ini adalah contoh *code* Python untuk mencoba *Boosting* pada dataset tersebut:

import pandas as pd

from sklearn.ensemble import AdaBoostClassifier

from sklearn.tree import DecisionTreeClassifier, plot\_tree

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import LabelEncoder, OrdinalEncoder

import matplotlib.pyplot as plt

# Read the CSV file

df = pd.read\_csv('drug200.csv')

# Separate the features and the target

X = df.drop('Drug', axis=1)

y = df['Drug']

# Initialize the OrdinalEncoder

ordinal\_encoder = OrdinalEncoder()

# Encode the features

X\_encoded = ordinal\_encoder.fit\_transform(X)

# Convert the string values of the target to numeric values

le = LabelEncoder()

y\_encoded = le.fit\_transform(y)

# Split the dataset into training and testing data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_encoded, y\_encoded, test\_size=0.2, random\_state=42)

# Use Decision Tree as the weak model

base\_model = DecisionTreeClassifier(max\_depth=1)

# Initialize the boosting model with 5 iterations

boosting\_model = AdaBoostClassifier(base\_model, n\_estimators=5, random\_state=42)

# Train the model on the training data

boosting\_model.fit(X\_train, y\_train)

# Make predictions on the testing data

y\_pred\_encoded = boosting\_model.predict(X\_test)

# Display the weight of the model at each iteration

for i, model in enumerate(boosting\_model.estimators\_):

    print(f"Iteration {i+1} - Model Weight: {boosting\_model.estimator\_weights\_[i]}")

# Calculate the accuracy of the model on the testing data

accuracy = accuracy\_score(y\_test, y\_pred\_encoded)

print(f"Model Accuracy: {accuracy}")

# Reverse encoding to display the prediction results in their original form

y\_test\_original = le.inverse\_transform(y\_test)

y\_pred\_original = le.inverse\_transform(y\_pred\_encoded)

# Visualize the Decision Tree at the first iteration

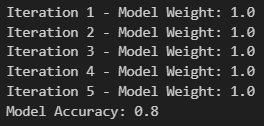
plt.figure(figsize=(10, 8))

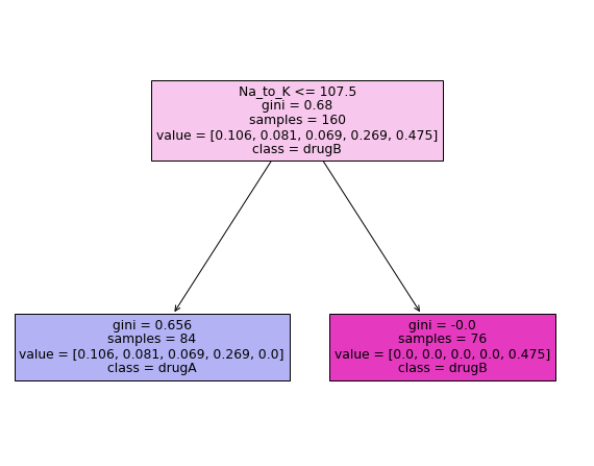
plot\_tree(boosting\_model.estimators\_[0], feature\_names=X.columns, class\_names=df['Drug'].unique(), filled=True)

plt.show()

1. Hasil *Boosting*

Berikut adalah hasil dari *code* Python diatas:





Berikut ini adalah beberapa kesimpulan yang dapat diambil dari hasil uji coba teknik Boosting:

* Jika nilai Na\_to\_K kurang dari atau sama dengan 107.5, maka hasil klasifikasi adalah drugA.
* Jika nilai Na\_to\_K lebih dari 107.5, maka hasil klasifikasi adalah drugB.

1. *Bagging*
2. Tujuan *Bagging*

Bagging, atau Bootstrap Aggregating, adalah teknik dalam data mining yang bertujuan untuk meningkatkan kestabilan dan kinerja model prediktif. Tujuannya adalah mengurangi varians model dengan membuat beberapa subset acak dari data pelatihan menggunakan metode bootstrap. Setiap subset digunakan untuk melatih model prediktif yang sama, dan hasil prediksi dari model-model tersebut digabungkan, meningkatkan ketepatan serta kehandalan model secara keseluruhan. Teknik ini terutama efektif dalam mengatasi overfitting dan meningkatkan konsistensi prediksi model.

1. Kekhususan *Bagging*

Kekhususan bagging terletak pada kemampuannya mengatasi overfitting dan meningkatkan konsistensi prediksi model. Dengan membuat beberapa subset acak dari data pelatihan dan melatih model pada masing-masing subset, bagging menghasilkan model yang lebih robust dan generalis. Bagging memiliki aplikabilitas luas, meningkatkan performa model pada berbagai jenis tugas prediktif dan menjadi pilihan utama dalam menghadapi kompleksitas dan variasi data.

1. Keuntungan *Bagging*

Keuntungan utama dari teknik bagging adalah peningkatan kestabilan dan kinerja model prediktif. Dengan membuat subset acak dari data pelatihan menggunakan metode bootstrap, bagging mengurangi varians model, sehingga model lebih cenderung memberikan prediksi yang konsisten dan dapat diandalkan pada data yang tidak terlihat sebelumnya. Selain itu, bagging efektif dalam mengatasi masalah overfitting, di mana model cenderung terlalu sesuai dengan data pelatihan dan kurang mampu menggeneralisasi pada data baru. Dengan melibatkan beberapa model yang dilatih pada subset berbeda, bagging memungkinkan model untuk memahami variasi yang lebih baik, menghasilkan solusi prediktif yang lebih baik untuk berbagai tugas dan jenis dataset.

1. Prinsip Kerja *Bagging*

Prinsip bagging, atau Bootstrap Aggregating, adalah memanfaatkan kekuatan diversifikasi model dengan melatih beberapa model prediktif pada subset acak dari data pelatihan. Dengan membuat subset menggunakan metode bootstrap, di mana setiap model diberi kesempatan melihat variasi unik dalam data, bagging mengurangi varians dan meningkatkan konsistensi prediksi keseluruhan. Dengan menggabungkan hasil prediksi dari model-model yang beragam, prinsip ini menghasilkan solusi prediktif yang lebih stabil dan dapat diandalkan, terutama efektif dalam menangani overfitting dan meningkatkan kemampuan generalisasi model pada data baru.

1. Uji Coba *Bagging*

Dataset yang akan digunakan dalam proyek ini adalah dataset untuk memprediksi performa dari mahasiswa. Untuk melihat lebih lengkap, dataset dapat diakses pada laman website <https://github.com/StevenO29/Tasks-Data-Mining/blob/main/drug200.csv>. Contoh dataset sebagai berikut:



Berikut ini adalah contoh *code* Python untuk mencoba *Boosting* pada dataset tersebut:

import pandas as pd

from sklearn.ensemble import BaggingClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

from sklearn.tree import plot\_tree

from sklearn.preprocessing import OrdinalEncoder

import matplotlib.pyplot as plt

df = pd.read\_csv('drug200.csv')

# Separate the features and the target

X = df.drop('Drug', axis=1)

y = df['Drug']

# Initialize the OrdinalEncoder

ordinal\_encoder = OrdinalEncoder()

# Encode the features

X\_encoded = ordinal\_encoder.fit\_transform(X)

# Split the dataset into training and testing data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_encoded, y, test\_size=0.2, random\_state=42)

# Use Decision Tree as the base estimator

base\_estimator = DecisionTreeClassifier(random\_state=42)

# Initialize the number of estimators

n\_estimators = 5

# Train the BaggingClassifier with a varying number of estimators

for i in range(1, n\_estimators + 1):

    bagging\_model = BaggingClassifier(base\_estimator=base\_estimator, n\_estimators=i, random\_state=42)

    bagging\_model.fit(X\_train, y\_train)

    y\_pred = bagging\_model.predict(X\_test)

    accuracy = accuracy\_score(y\_test, y\_pred)

    print(f"Iteration {i} - Model Accuracy: {accuracy}")

# Calculate the accuracy score

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Model Accuracy: {accuracy:.4f}')

# Get the unique class names

class\_names = df['Drug'].unique()

# Visualize the Decision Trees in the ensemble

for i in range(n\_estimators):

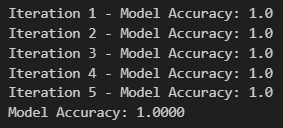
    plt.figure(figsize=(12, 8))

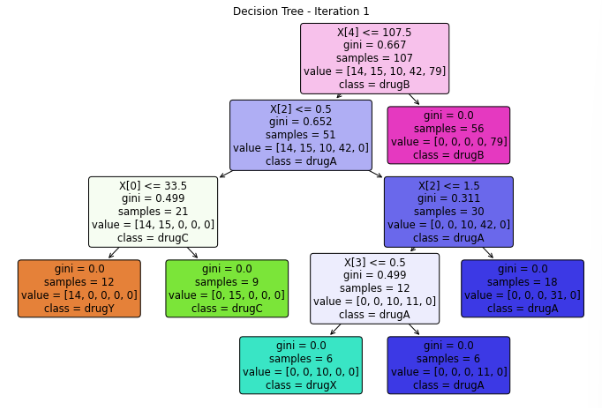
    plot\_tree(bagging\_model.estimators\_[i], filled=True, rounded=True, class\_names=class\_names)

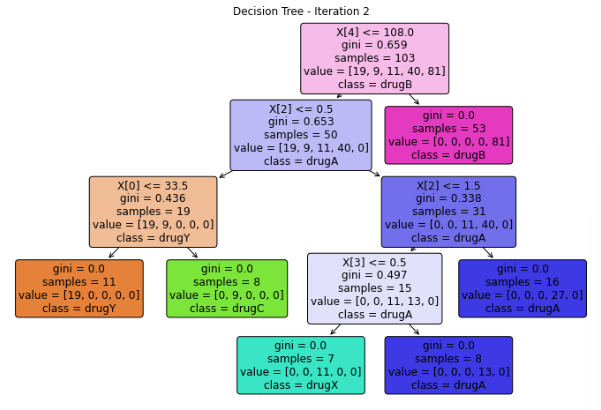
    plt.title(f'Decision Tree - Iteration {i + 1}')

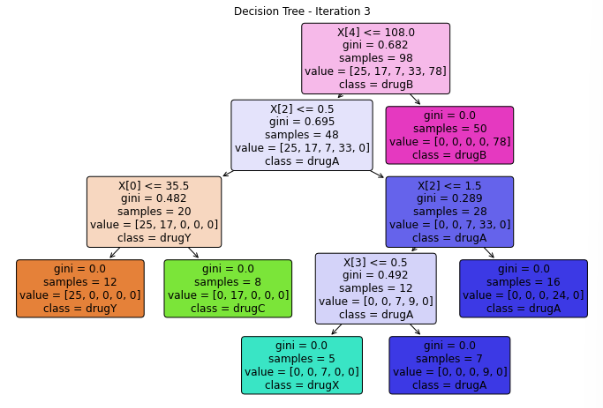
    plt.show()

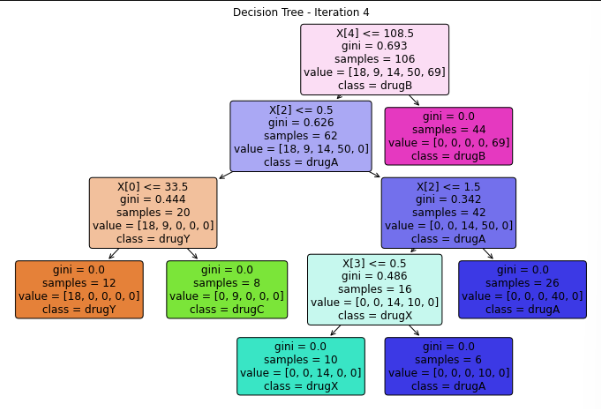
1. Hasil *Bagging*

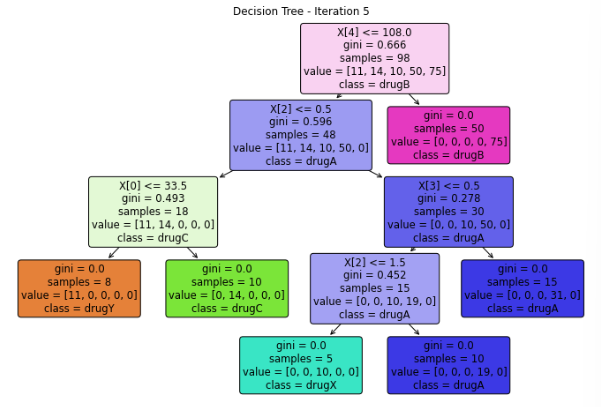












Berikut adalah beberapa kesimpulan yang dapat diambil dari visualisasi Decision Tree – Iteration 5:

* Pada baris pertama, jika nilai X[4] kurang dari atau sama dengan 108.0, hasil klasifikasi adalah drugB.
* Jika kondisi pada baris pertama tidak terpenuhi (nilai X[4] lebih dari 108.0), kita beralih ke baris kedua:
* Jika nilai X[2] kurang dari atau sama dengan 0.5, hasil klasifikasi adalah drugA.
* Jika nilai X[2] lebih dari 0.5, hasil klasifikasi adalah drugB (kondisi dari baris kedua tidak diperhatikan karena kita sudah masuk ke kanan pada baris pertama).
* Jika pada baris kedua kita mengambil cabang kiri (nilai X[2] kurang dari atau sama dengan 0.5), kita beralih ke baris ketiga:
* Jika nilai X[0] kurang dari atau sama dengan 33.5, hasil klasifikasi adalah drugC.
* Jika nilai X[0] lebih dari 33.5, kita beralih ke kanan:
* Jika nilai X[3] kurang dari atau sama dengan 0.5, hasil klasifikasi adalah drugA.
* Jika nilai X[3] lebih dari 0.5, hasil klasifikasi adalah drugY.
* Jika pada baris ketiga kita mengambil cabang kanan (nilai X[2] lebih dari 0.5), kita beralih ke baris keempat:
* Jika nilai X[2] kurang dari atau sama dengan 1.5, hasil klasifikasi adalah drugA.
* Jika nilai X[2] lebih dari 1.5, kita beralih ke kanan:
* Jika nilai X[3] kurang dari atau sama dengan 0.5, hasil klasifikasi adalah drugX.
* Jika nilai X[3] lebih dari 0.5, hasil klasifikasi adalah drugA.
* Jika pada baris keempat kita mengambil cabang kanan (nilai X[2] lebih dari 1.5), hasil klasifikasi adalah drugA.
* Jika pada baris kelima kita mengambil cabang kiri, hasil klasifikasi adalah drugX.
* Jika pada baris kelima kita mengambil cabang kanan, hasil klasifikasi adalah drugA.

1. *Random Forest*
2. Tujuan *Random Forest*

Random Forest adalah sebuah algoritma dalam machine learning yang menggunakan konsep ensemble learning untuk meningkatkan kinerja prediksi. Algoritma ini membangun sejumlah pohon keputusan selama proses pelatihan dan menggabungkan hasil prediksi dari setiap pohon untuk menghasilkan prediksi akhir. Tujuan utama Random Forest dalam data mining adalah untuk meningkatkan akurasi dan mengurangi overfitting. Dengan kombinasi beberapa pohon keputusan yang dibangun secara acak, Random Forest dapat mengatasi kelemahan individual pohon keputusan dan memberikan hasil yang lebih stabil dan dapat diandalkan dalam tugas klasifikasi dan regresi.

1. Kekhususan *Random Forest*

Kekhususan utama dari Random Forest terletak pada pendekatan ensemble learning-nya, di mana algoritma ini membangun sejumlah besar pohon keputusan secara acak dan menggabungkan hasil prediksi mereka. Hal ini membuat Random Forest sangat tahan terhadap overfitting dan mampu memberikan prediksi yang konsisten serta akurat. Kemampuannya menangani masalah multikolinearitas, data yang hilang, dan implementasinya yang relatif sederhana membuatnya menjadi pilihan yang populer dalam berbagai tugas machine learning, terutama di lingkungan dengan data kompleks dan variasi tinggi.

1. Keuntungan *Random Forest*

Keuntungan utama dari Random Forest adalah kemampuannya untuk menghasilkan prediksi yang akurat dan stabil dalam berbagai tugas machine learning. Dengan menggabungkan hasil dari banyak pohon keputusan yang dibangun secara acak, algoritma ini mengurangi risiko overfitting dan memberikan ketangguhan terhadap variasi dalam data. Selain itu, Random Forest dapat menangani masalah multikolinearitas, data yang hilang, dan fitur-fitur yang tidak relevan, membuatnya sangat adaptif untuk dataset yang kompleks. Fleksibilitas implementasi, kemampuan paralelisme, dan relatif sedikit parameter yang perlu dikonfigurasi juga menjadikan Random Forest pilihan yang efisien dan kuat dalam aplikasi praktis.

1. Prinsip Kerja *Random Forest*

Prinsip kerja Random Forest didasarkan pada konsep ensemble learning, di mana algoritma ini membangun sejumlah besar pohon keputusan secara acak selama fase pelatihan. Setiap pohon dibangun dengan menggunakan subset acak dari data pelatihan dan subset acak dari fitur. Selama prediksi, hasil dari setiap pohon digabungkan untuk membentuk prediksi akhir. Dengan menggabungkan banyak pohon yang dibangun secara acak, Random Forest dapat mengurangi overfitting dan meningkatkan stabilitas serta akurasi prediksi. Seleksi acak fitur dan data juga memberikan ketahanan terhadap variasi dalam dataset, menjadikan Random Forest efektif dalam berbagai tugas machine learning.

1. Uji Coba *Random Forest*

Dataset yang akan digunakan dalam proyek ini adalah dataset untuk memprediksi performa dari mahasiswa. Untuk melihat lebih lengkap, dataset dapat diakses pada laman website <https://github.com/StevenO29/Tasks-Data-Mining/blob/main/drug200.csv>. Contoh dataset sebagai berikut:

1. 

Berikut ini adalah contoh *code* Python untuk mencoba *Random Forest* pada dataset tersebut:

import pandas as pd

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

from sklearn.tree import plot\_tree

from sklearn.preprocessing import OrdinalEncoder

import matplotlib.pyplot as plt

df = pd.read\_csv('drug200.csv')

# Separate the features and the target

X = df.drop('Drug', axis=1)

y = df['Drug']

# Initialize the OrdinalEncoder

ordinal\_encoder = OrdinalEncoder()

# Encode the features

X\_encoded = ordinal\_encoder.fit\_transform(X)

# Split the dataset into training and testing data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_encoded, y, test\_size=0.2, random\_state=42)

# Use Random Forest with 4 trees (estimators)

random\_forest\_model = RandomForestClassifier(n\_estimators=4, random\_state=42)

# Train the model on the training data

random\_forest\_model.fit(X\_train, y\_train)

# Make predictions on the testing data

y\_pred = random\_forest\_model.predict(X\_test)

# Visualize the Decision Trees in the ensemble

for i, tree\_in\_forest in enumerate(random\_forest\_model.estimators\_):

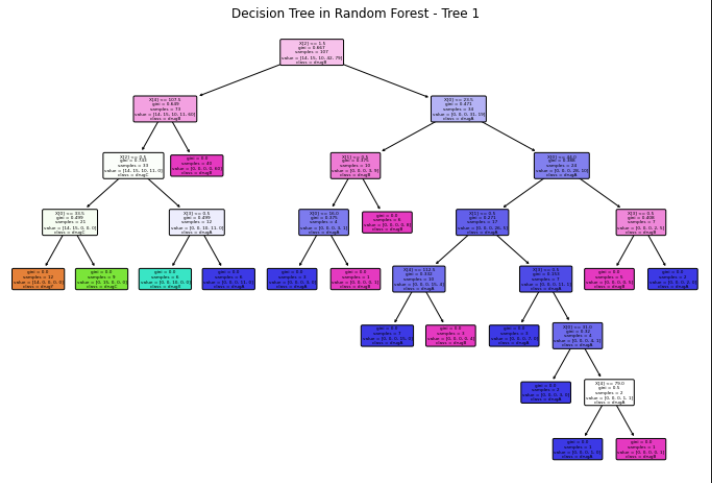
    plt.figure(figsize=(12, 8))

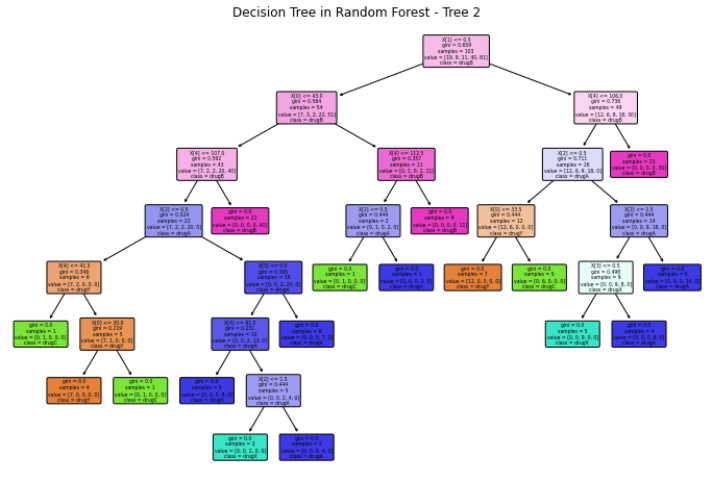
    plot\_tree(tree\_in\_forest, filled=True, rounded=True)

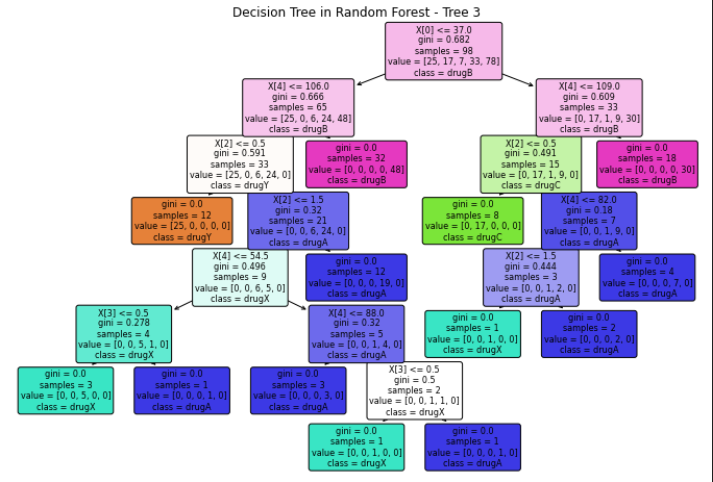
    plt.title(f'Decision Tree in Random Forest - Tree {i + 1}')

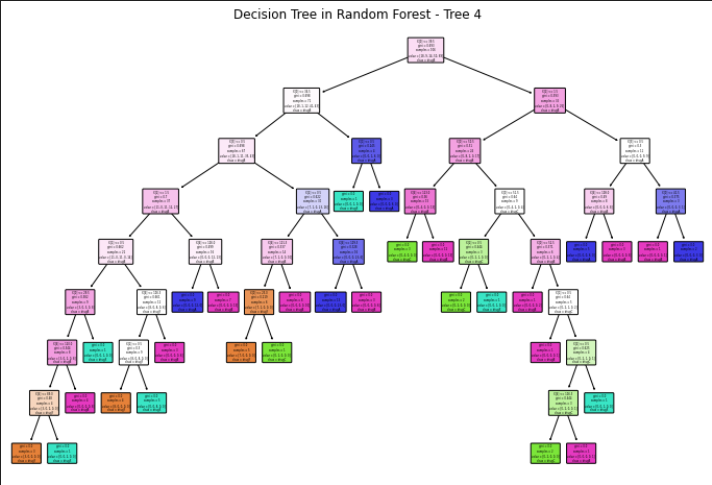
    plt.show()

1. Hasil *Random Forest*









Berikut adalah beberapa kesimpulan yang dapat diambil dari visualisasi Decision Tree in Random Forest – Tree 3:

* Pada baris pertama, jika nilai X[0] kurang dari atau sama dengan 37.0, hasil klasifikasi adalah drugB.
* Jika pada baris pertama kita mengambil cabang kanan (nilai X[0] lebih dari 37.0), kita beralih ke baris kedua:
* Jika nilai X[4] kurang dari atau sama dengan 106.0, hasil klasifikasi adalah drugB.
* Jika nilai X[4] lebih dari 106.0 dan kurang dari atau sama dengan 109.0, kita beralih ke kanan pada baris kedua, dan hasil klasifikasi adalah drugB.
* Jika nilai X[4] lebih dari 109.0, kita beralih ke baris ketiga:
* Jika nilai X[2] kurang dari atau sama dengan 0.5, hasil klasifikasi adalah drugY.
* Jika nilai X[2] lebih dari 0.5 dan kurang dari atau sama dengan 1.5, kita beralih ke kanan pada baris ketiga, dan hasil klasifikasi adalah drugC.
* Jika nilai X[2] lebih dari 1.5, kita beralih ke baris keempat:
* Jika nilai X[4] kurang dari atau sama dengan 82.0, hasil klasifikasi adalah drugA.
* Jika nilai X[4] lebih dari 82.0, kita beralih ke paling kanan pada baris keempat, dan hasil klasifikasi adalah drugA.
* Jika pada baris ketiga kita mengambil cabang kiri (nilai X[2] kurang dari atau sama dengan 0.5), hasil klasifikasi adalah drugY.
* Jika pada baris ketiga kita mengambil cabang kanan (nilai X[2] lebih dari 0.5), kita beralih ke paling kanan pada baris ketiga, dan hasil klasifikasi adalah drugB.
* Jika pada baris keempat kita mengambil cabang kiri (nilai X[2] kurang dari atau sama dengan 1.5), kita beralih ke paling kiri pada baris keempat, dan hasil klasifikasi adalah drugY.
* Jika pada baris keempat kita mengambil cabang kanan (nilai X[2] lebih dari 1.5), kita beralih ke kanan pada baris keempat:
* Jika nilai X[4] kurang dari atau sama dengan 88.0, hasil klasifikasi adalah drugA.
* Jika nilai X[4] lebih dari 88.0, kita beralih ke paling kanan pada baris keempat, dan hasil klasifikasi adalah drugA.
* Jika pada baris kelima kita mengambil cabang kiri (nilai X[4] kurang dari atau sama dengan 54.5), kita beralih ke paling kiri pada baris kelima, dan hasil klasifikasi adalah drugX.
* Jika pada baris kelima kita mengambil cabang kanan (nilai X[4] lebih dari 54.5 dan kurang dari atau sama dengan 106.0), kita beralih ke kiri pada baris kelima, dan hasil klasifikasi adalah drugA.
* Jika pada baris kelima kita mengambil cabang kanan (nilai X[4] lebih dari 106.0 dan kurang dari atau sama dengan 109.0), kita beralih ke kanan pada baris kelima:
* Jika nilai X[2] kurang dari atau sama dengan 1.5, hasil klasifikasi adalah drugA.
* Jika nilai X[2] lebih dari 1.5, kita beralih ke paling kanan pada baris kelima, dan hasil klasifikasi adalah drugA.
* Jika pada baris keenam kita mengambil cabang kiri (nilai X[3] kurang dari atau sama dengan 0.5), kita beralih ke paling kiri pada baris keenam:
* Jika pada baris keenam kita mengambil cabang kiri (nilai X[4] kurang dari atau sama dengan 88.0), hasil klasifikasi adalah drugA.
* Jika pada baris keenam kita mengambil cabang kanan (nilai X[4] lebih dari 88.0), kita beralih ke paling kanan pada baris keenam.
* Jika pada baris ketujuh kita mengambil cabang kiri (nilai X[4] kurang dari atau sama dengan 54.5), kita beralih ke paling kiri pada baris ketujuh:
* Jika pada baris ketujuh kita mengambil cabang kiri (nilai X[3] kurang dari atau sama dengan 0.5), hasil klasifikasi adalah drugX.
* Jika pada baris ketujuh kita mengambil cabang kanan (nilai X[3] lebih dari 0.5), kita beralih ke paling kanan pada baris ketujuh.
* Jika pada baris kedelapan kita mengambil cabang kiri, hasil klasifikasi adalah drugX.
* Jika pada baris kedelapan kita mengambil cabang kanan, hasil klasifikasi adalah drugA.

1. *Regression Tree*
2. Tujuan *Regression Tree*

Regression Tree, atau yang dikenal sebagai pohon regresi, adalah algoritma pembelajaran mesin dalam data mining yang digunakan untuk memodelkan hubungan antara fitur masukan dan variabel target yang bersifat kontinu. Algoritma ini membangun struktur pohon di mana setiap simpul internal mewakili keputusan berdasarkan suatu fitur, dan setiap simpul daun berisi nilai target yang diprediksi. Tujuan dari regresi pohon adalah untuk membagi data secara rekursif menjadi subset dengan meminimalkan varians dari variabel target dalam setiap subset. Dengan membuat prediksi berdasarkan nilai rata-rata variabel target di setiap simpul daun, regresi pohon memberikan cara yang sederhana dan mudah diinterpretasikan untuk menangkap hubungan kompleks dalam data, menjadikannya alat berharga untuk pemodelan prediktif dan analisis.

1. Kekhususan *Regression Tree*

Keunggulan khusus Regression Tree terletak pada kemampuannya untuk menangkap dan merepresentasikan hubungan non-linier dan kompleks dalam data secara intuitif. Dengan struktur pohon yang terdiri dari keputusan simpel berbasis fitur, regresi pohon memudahkan interpretasi dan pemahaman model. Selain itu, algoritma ini dapat menangani data yang memiliki interaksi variabel yang kompleks, serta toleran terhadap adanya pencilan (outliers) dalam dataset. Fleksibilitas dan kemampuan regresi pohon membuatnya menjadi pilihan yang populer dalam pemodelan prediktif, terutama ketika hubungan antara variabel input dan output tidak linear dan rumit.

1. Keuntungan *Regression Tree*

Keuntungan utama dari regresi pohon melibatkan kemampuan untuk mengatasi kompleksitas dan non-linieritas dalam data, serta menyediakan interpretasi yang mudah dimengerti. Pohon keputusan dapat memodelkan hubungan yang rumit dan memetakan pola yang sulit dikenali oleh model statistik tradisional. Selain itu, regresi pohon dapat menangani campuran tipe data (numerik dan kategorikal) tanpa memerlukan pra-pemrosesan yang rumit. Kemampuan untuk menangani data yang tidak terstruktur dan memberikan hasil yang mudah diinterpretasikan membuat regresi pohon menjadi pilihan yang efektif dalam berbagai aplikasi, termasuk prediksi, analisis risiko, dan pengambilan keputusan.

1. Prinsip Kerja *Regression Tree*

Prinsip kerja regresi pohon melibatkan proses pembentukan struktur pohon berdasarkan pemilihan fitur yang paling informatif untuk membagi data. Pohon ini dibangun secara rekursif dengan memilih aturan pemisahan di setiap simpul berdasarkan kriteria seperti mean squared error. Pada setiap langkah, algoritma mencari pemisahan yang menghasilkan subset data dengan varians target yang minimal. Proses ini terus berlanjut hingga mencapai kriteria penghentian, seperti kedalaman pohon maksimum atau jumlah sampel minimum di setiap simpul. Setelah pohon terbentuk, prediksi untuk suatu instance data dilakukan dengan menelusuri pohon dari akar hingga daun dan mengambil nilai rata-rata di daun tersebut. Prinsip ini memungkinkan regresi pohon menangkap hubungan yang kompleks dan memberikan prediksi yang sesuai dengan struktur data yang diberikan.

1. Uji Coba *Regression Tree*

Dataset yang akan digunakan dalam proyek ini adalah dataset untuk memprediksi performa dari mahasiswa. Untuk melihat lebih lengkap, dataset dapat diakses pada laman website <https://github.com/StevenO29/Tasks-Data-Mining/blob/main/drug200.csv>. Contoh dataset sebagai berikut:

1. 

Berikut ini adalah contoh *code* Python untuk mencoba *Regression Tree* pada dataset tersebut:

import pandas as pd

from sklearn.tree import DecisionTreeRegressor, plot\_tree

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

from sklearn.preprocessing import LabelEncoder, OrdinalEncoder

import matplotlib.pyplot as plt

# Load your dataset

df = pd.read\_csv('drug200.csv')

# Initialize the LabelEncoder

label\_encoder = LabelEncoder()

# Convert the 'Drug' column to numeric values

df['Drug'] = label\_encoder.fit\_transform(df['Drug'])

# Choose one feature as the target variable (for example, 'Drug')

X = df.drop(['Drug'], axis=1)

y = df['Drug']

# Initialize the OrdinalEncoder

ordinal\_encoder = OrdinalEncoder()

# Encode the features

X\_encoded = ordinal\_encoder.fit\_transform(X)

# Split the dataset into training and testing data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_encoded, y, test\_size=0.2, random\_state=42)

# Initialize the Decision Tree Regressor

regression\_tree = DecisionTreeRegressor(max\_depth=None, random\_state=42)

# Train the model on the training data

regression\_tree.fit(X\_train, y\_train)

# Prune by setting the max\_depth parameter

regression\_tree\_pruned = DecisionTreeRegressor(max\_depth=3, random\_state=42)

regression\_tree\_pruned.fit(X\_train, y\_train)

# Print the MSE on the testing data after pruning

y\_pred\_train\_pruned = regression\_tree\_pruned.predict(X\_train)

mse\_train\_pruned = mean\_squared\_error(y\_train, y\_pred\_train\_pruned)

print(f'MSE on Training Data after Pruning: {mse\_train\_pruned:.4f}')

# Visualize the Decision Tree after pruning

plt.figure(figsize=(15, 10))

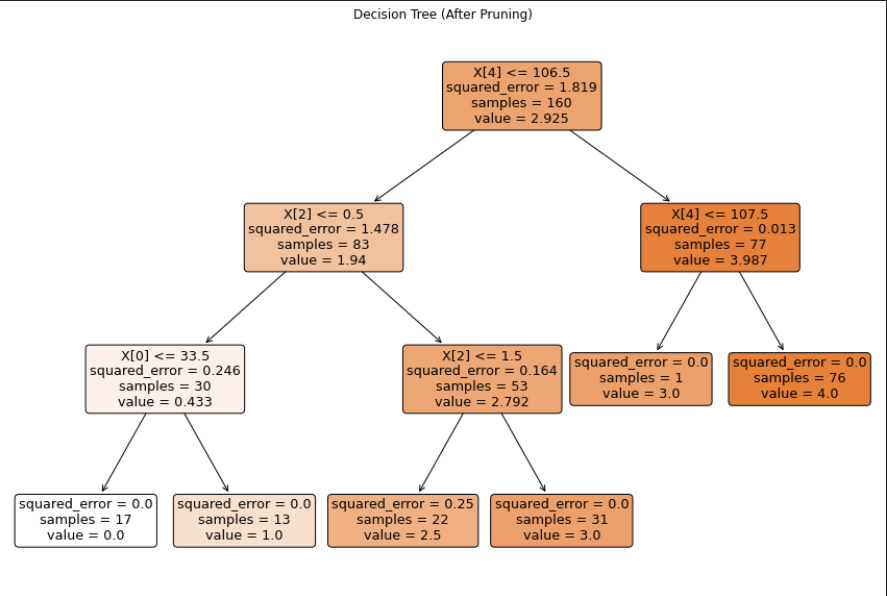
plot\_tree(regression\_tree\_pruned, filled=True, rounded=True)

plt.title('Decision Tree (After Pruning)')

plt.show()

1. Hasil *Regression Tree*





MSE (Mean Squared Error) pada Training Data setelah dilakukan proses pruning sebesar 0.0344 mengindikasikan bahwa model regression tree yang telah diubah atau disederhanakan memiliki tingkat kesalahan yang relatif rendah dalam melakukan prediksi terhadap data pelatihan. Proses pruning pada regression tree bertujuan untuk mengurangi kompleksitas model, mencegah overfitting, dan meningkatkan generalisasi terhadap data baru. Dengan MSE yang rendah, dapat disimpulkan bahwa model yang telah dihasilkan setelah pruning mampu memberikan estimasi yang akurat terhadap variabel target pada data latih, sehingga lebih dapat diandalkan untuk menggeneralisasi pola-pola yang ada dalam dataset yang belum pernah dilihat sebelumnya.