



# **Proses Perhitungan Decision Tree dan K-Nearest Neighbors Menggunakan Google Colab**

**Oleh: Kelompok FORKA**

# Anggota FORKA



**Siti Kamila**



**Retno Ayu  
AmbarSari**



**Dwi Okta  
Ramadani**



**Faiqotul Jannah**



**Alda Zaneta  
Salsabila**



# Studi Kasus 1

## Klasifikasi Jenis Mobil Menggunakan Decision Tree



# Apa itu Decision Tree?

## Pengertian:

**Metode analisis yang digunakan untuk mengklasifikasi atau memprediksi sesuatu berdasarkan pola dari data. Strukturnya menyerupai pohon:**

- 1. Root (akar)** → atribut paling penting
- 2. Branch (cabang)** → keputusan/percabangan
- 3. Leaf (daun)** → hasil akhir/prediksi

## Cara Kerja Decision Tree

- 1 Memilih atribut yang paling berpengaruh menggunakan perhitungan seperti Entropy dan Information Gain.
- 2 Memecah data berdasarkan atribut tersebut.
- 2 Proses berulang sampai terbentuk hasil akhir (kelas).

## Tujuan Decision Tree

1. Membantu pengambilan keputusan secara otomatis dari data.
2. Melakukan klasifikasi, misalnya: rajin / tidak rajin, lulus / tidak lulus, loyal / tidak loyal.
3. Melakukan prediksi berdasarkan pola data.
4. Mengetahui faktor paling berpengaruh pada suatu keputusan.
5. Menyajikan hasil dalam bentuk visual yang mudah dipahami.



# Proses Perhitungan

```
import pandas as pd
```

## Penjelasan

Kode ini digunakan untuk mengimpor library pandas, yang dipakai untuk membaca, mengolah, dan menganalisis data, karena tanpa pandas, file CSV atau DataFrame tidak bisa dibaca atau diolah dengan mudah.

## Filosofi Cerita

Ini seperti kamu membuka buku data mobil yang berisi spesifikasi mobil-mobil yang ingin kamu gunakan untuk melatih “sales mobil” barumu (komputer).

# Proses Perhitungan

```
from google.colab import files  
uploaded = files.upload()  
  
Choose File's decisiontree_1 (1).csv  
decisiontree_1 (1).csv(text/csv) - 927 bytes, last modified: 11/29/2025 - 100% done  
Saving decisiontree_1 (1).csv to decisiontree_1 (1) (5).csv
```

## Penjelasan

Kode ini digunakan untuk **mengunggah file CSV dari komputer ke Google Colab**, dimana `files.upload()` akan menampilkan kotak dialog agar kita bisa memilih file, dan hasil upload disimpan di variabel `uploaded` untuk diproses selanjutnya.

## Filosofi Cerita

Ini seperti kamu memberikan daftar mobil (dataset) ke sales baru tersebut supaya dia bisa mulai belajar. Tanpa file ini, dia tidak punya data apa pun untuk dipahami.



# Proses Perhitungan

```
data = pd.read_csv('decisiontree_1 (1).csv')

encoding = {
    "mesin": {"bensin": 0, "diesel": 1},
    "penggerak": {"depan": 0, "belakang": 1}
}

data.replace(encoding, inplace=True)
data.head()
```

/tmp/ipython-input-3398562086.py:8: FutureWarning:  
data.replace(encoding, inplace=True)

ID	mesin	bangku	penggerak	label	
0	1	0	4	0	sedan
1	2	0	2	0	sedan
2	3	0	8	1	minibus
3	4	1	6	0	minibus
4	5	0	5	1	minibus

## Penjelasan

- Membaca data mobil dari file CSV.
- Mengubah tulisan menjadi angka (encoding).
- Menampilkan lima baris pertama data.

## Filosofi Cerita

Ini seperti kamu mengubah istilah-istilah tentang mobil seperti “bensin”, “diesel”, dan “penggerak depan” menjadi angka, supaya si sales (komputer) yang masih belajar bisa memahami datanya dengan lebih mudah. Karena komputer tidak paham kata-kata, dia hanya bisa belajar kalau informasinya sudah diubah ke bentuk angka.



# Proses Perhitungan

## Penjelasan

Kode ini hanya menampilkan 5 baris pertama dari DataFrame untuk mengecek data, terutama setelah proses encoding agar kita yakin nilainya sudah benar.

## Filosofi Cerita

Sama seperti sales mobil mengecek beberapa contoh mobil dulu dari daftar untuk memastikan dia benar-benar memahami bentuk datanya.

```
data.head()
```

ID	mesin	bangku	penggerak	label	
0	4	0	0	sedan	
1	2	0	2	0	sedan
2	3	0	8	1	minibus
3	4	1	6	0	minibus
4	5	0	5	1	minibus

# Proses Perhitungan

```
x = data.drop(['ID', 'label'], axis =1)  
y = data['label']
```

## Penjelasan

Kode ini memisahkan kolom fitur x yang terdiri dari mesin, bangku, dan penggerak untuk dipakai model, sedangkan y adalah kolom target label (sedan atau minibus) yang ingin diprediksi, dan kolom ID dibuang karena tidak relevan untuk prediksi.

## Filosofi Cerita

Ini seperti kamu menjelaskan ke sales:

“Yang penting kamu lihat mesin, bangku, dan penggerak untuk menilai mobil. Kolom ID nggak perlu, yang mau kamu tebak adalah tipe mobilnya, yaitu sedan atau minibus.”



```
import sklearn.model_selection as ms

x_train, x_test, y_train, y_test = ms.train_test_split(x,
                                                       y,
                                                       test_size=0.5, random_state=42)
```

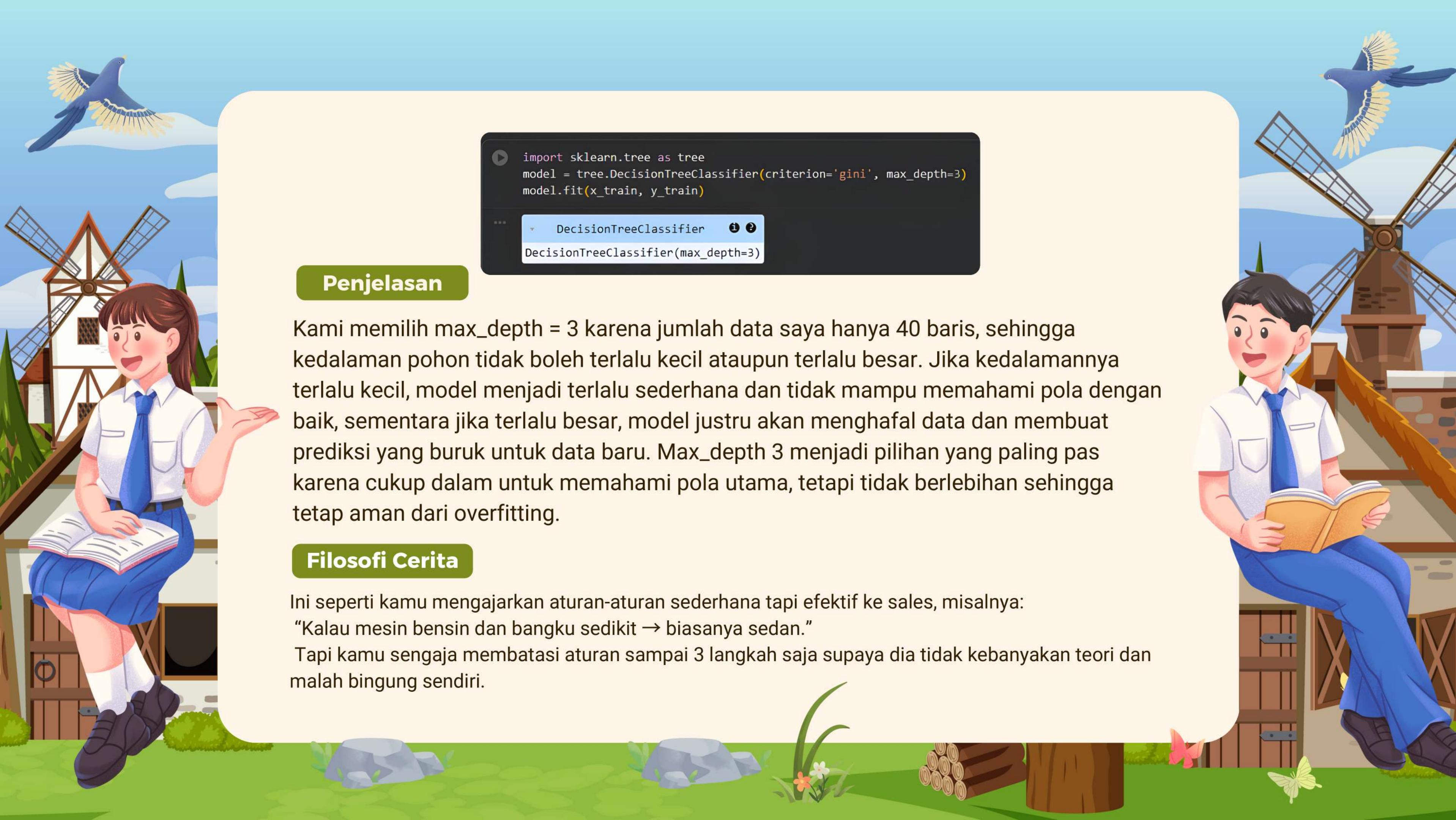
## Penjelasan

- Pada Dataset ada 40 baris. Dengan test\_size=0.5, maka:
  - 20 baris → training set
  - 20 baris → testing set
- **Alasan pilih 0.5:** Karena ukuran dataset kita kecil (40 baris), Kalau dataset kecil, kamu harus seimbang dalam memberi cukup data untuk model belajar, dan memberi cukup data untuk menguji model secara adil. Dan pembagian 50%–50% adalah titik yang paling aman untuk dataset kecil.
- Random\_state= 42 → memastikan pembagian 20/20 sama setiap dijalankan. Tanpa ini, hasil akurasi bisa berbeda tiap run.

Jadi angka 0.5 dan 42 dipilih sesuai ukuran dataset supaya model belajar cukup dan bisa diuji secara representatif.

## Filosofi Cerita

Ini seperti kamu membagi 40 mobil menjadi dua kelompok:  
20 mobil untuk melatih sales mengenali tipe mobil,  
dan 20 mobil lagi untuk mengetes apakah dia sudah paham.  
Pembagian yang seimbang bikin latihannya adil dan hasil tesnya akurat.



```
import sklearn.tree as tree
model = tree.DecisionTreeClassifier(criterion='gini', max_depth=3)
model.fit(x_train, y_train)

...
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=3)
```

### Penjelasan

Kami memilih `max_depth = 3` karena jumlah data saya hanya 40 baris, sehingga kedalaman pohon tidak boleh terlalu kecil ataupun terlalu besar. Jika kedalamannya terlalu kecil, model menjadi terlalu sederhana dan tidak mampu memahami pola dengan baik, sementara jika terlalu besar, model justru akan menghafal data dan membuat prediksi yang buruk untuk data baru. `Max_depth 3` menjadi pilihan yang paling pas karena cukup dalam untuk memahami pola utama, tetapi tidak berlebihan sehingga tetap aman dari overfitting.

### Filosofi Cerita

Ini seperti kamu mengajarkan aturan-aturan sederhana tapi efektif ke sales, misalnya:

“Kalau mesin bensin dan bangku sedikit → biasanya sedan.”

Tapi kamu sengaja membatasi aturan sampai 3 langkah saja supaya dia tidak kebanyakan teori dan malah bingung sendiri.

# Proses Perhitungan

## Penjelasan

Kode ini digunakan untuk memprediksi label data testing dengan model yang sudah dilatih, dan hasil prediksi ditampilkan dalam bentuk array, dimana setiap elemen array adalah prediksi label untuk satu baris data testing.

## Filosofi Cerita

Ini saatnya sales mobil dites. Kamu kasih mobil baru dan dia harus menebak:  
“Ini sedan atau minibus?”

Hasil tebakannya ditampilkan sebagai list jawaban.

```
y_prediksi = model.predict(x_test)  
y_prediksi  
  
array(['minibus', 'sedan', 'minibus', 'sedan', 'minibus', 'sedan',  
       'minibus', 'sedan', 'minibus', 'sedan', 'minibus', 'minibus',  
       'minibus', 'sedan', 'minibus', 'sedan', 'sedan'], dtype=object)
```

# Proses Perhitungan

```
import sklearn.metrics as met  
print (met.accuracy_score(y_test, y_prediksi))  
  
0.9411764705882353
```

## Penjelasan

**Kode ini menghitung akurasi model dengan membandingkan prediksi `y_prediksi` dengan label asli `y_test`, dan hasil `0.941176` berarti model berhasil memprediksi dengan tingkat kebenaran sekitar 94%.**

## Filosofi Cerita

Ini seperti kamu menghitung berapa banyak tebakan si sales yang benar.

Kalau akurasinya 94% berarti sales-mu hampir selalu bisa membedakan sedan dan minibus dengan tepat.

# Proses Perhitungan

```
import matplotlib.pyplot as plt  
from sklearn import tree  
  
plt.subplots(figsize=(10,10))  
tree.plot_tree(model, fontsize=10)  
plt.show()
```

## Filosofi Cerita

Ini seperti menggambar mind map cara berpikir sales tersebut.

Kamu bisa melihat langkah-langkah logikanya:  
“Kalau mesin begini → lihat bangku → lalu lihat penggerak → hasilnya sedan/minibus.”

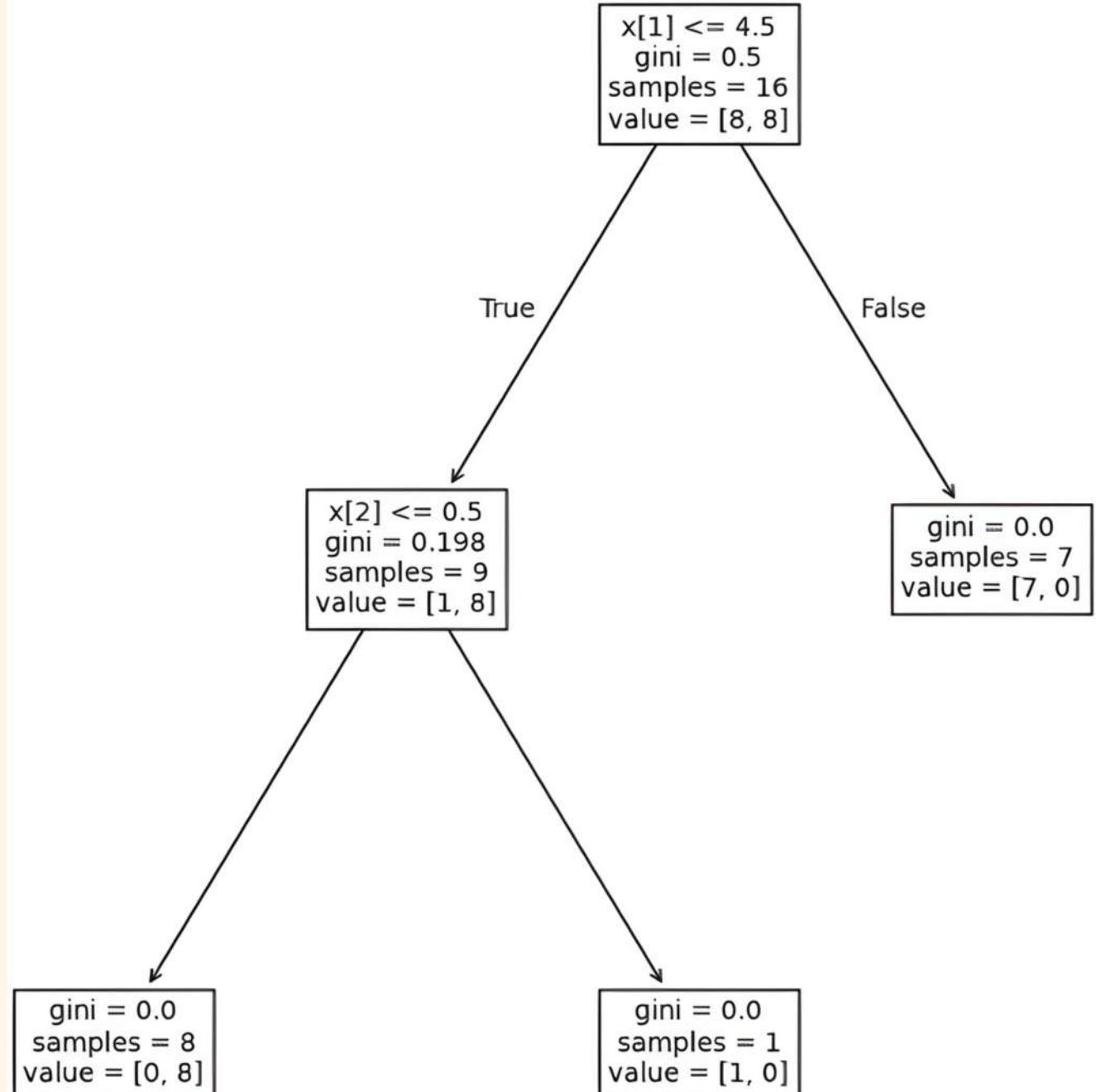
Jadi kamu bisa memahami kenapa dia mengambil keputusan tersebut.

## Penjelasan

Kode ini digunakan untuk memvisualisasikan decision tree agar logika keputusan model mudah dipahami, dimana `figsize=(10,10)` membuat gambar cukup besar dan jelas serta `fontsize=10` membuat teks di diagram mudah dibaca, sehingga kita bisa melihat bagaimana model membagi data pada setiap node dan apa hasil prediksi di tiap leaf.



# Hasil Pohon Keputusan



- Node paling atas adalah node paling tidak murni, karena datanya masih bercampur seimbang antara dua kelas. Ini menunjukkan model belum bisa menebak kelas secara yakin.
- Setiap percabangan (split) dibuat berdasarkan kondisi tertentu, misalnya  $x[1] \leq 4.5$ , untuk memisahkan data agar menjadi lebih murni.
- Semakin ke bawah, node akan semakin murni, yaitu isinya cenderung hanya satu kelas saja.
- Jumlah sampel menunjukkan berapa data yang berada di node tersebut.
- Value memperlihatkan distribusi kelas (misalnya [sedan, minibus]) sehingga kita tahu kelas mana yang dominan.
- Gini menunjukkan tingkat ketidakmurnian:
  - 1.) Semakin tinggi, node semakin campur.
  - 2.) Semakin rendah, node semakin bersih dan dekat ke keputusan akhir.
- Node daun (leaf) adalah node yang sudah murni, di mana model menetapkan prediksi berdasarkan kelas mayoritas.

# Studi Kasus 2

## Rekomendasi Film Menggunakan K-Nearest Neighbors



# Apa itu KNN?

## Pengertian:

**"KNN (K-Nearest Neighbors) adalah metode yang memprediksi sesuatu dengan cara melihat data-data terdekat yang mirip, lalu mengambil keputusan berdasarkan mayoritas dari tetangga terdekat tersebut."**

## Tujuan KNN

- Mengklasifikasikan data baru ke dalam kelompok yang paling sesuai berdasarkan kemiripan.
- Memprediksi nilai atau kategori suatu data dengan melihat mayoritas tetangga terdekatnya.
- Mengenali pola hubungan antar data menggunakan perhitungan jarak yang sederhana.
- Memberikan hasil analisis yang fleksibel dan bisa langsung menyesuaikan jika ada data baru masuk.

## Cara Kerja KNN

1. **Tentukan Angka K:** Pilih jumlah tetangga terdekat (angka K) yang akan dijadikan acuan pengambilan keputusan.
2. **Hitung Jarak:** Ukur jarak antara data baru yang ingin dicek dengan semua data yang sudah tersimpan di database. Urutkan Tetangga: Urutkan data dari jarak yang paling dekat (terkecil) ke yang paling jauh.
3. **Ambil Mayoritas:** Lihat kategori apa yang paling banyak mendominasi di antara sejumlah K tetangga terdekat tadi.
4. **Hasil Akhir:** Tetapkan kategori data baru tersebut mengikuti kategori mayoritas tetangganya.



```
import pandas as pd
import numpy as np
#pandas buat baca data, buat tabel, bersihin data, gabungin data (alat ngolah data)
#numpy library buat ngulah angka dan operasi mtk
```

### Penjelasan

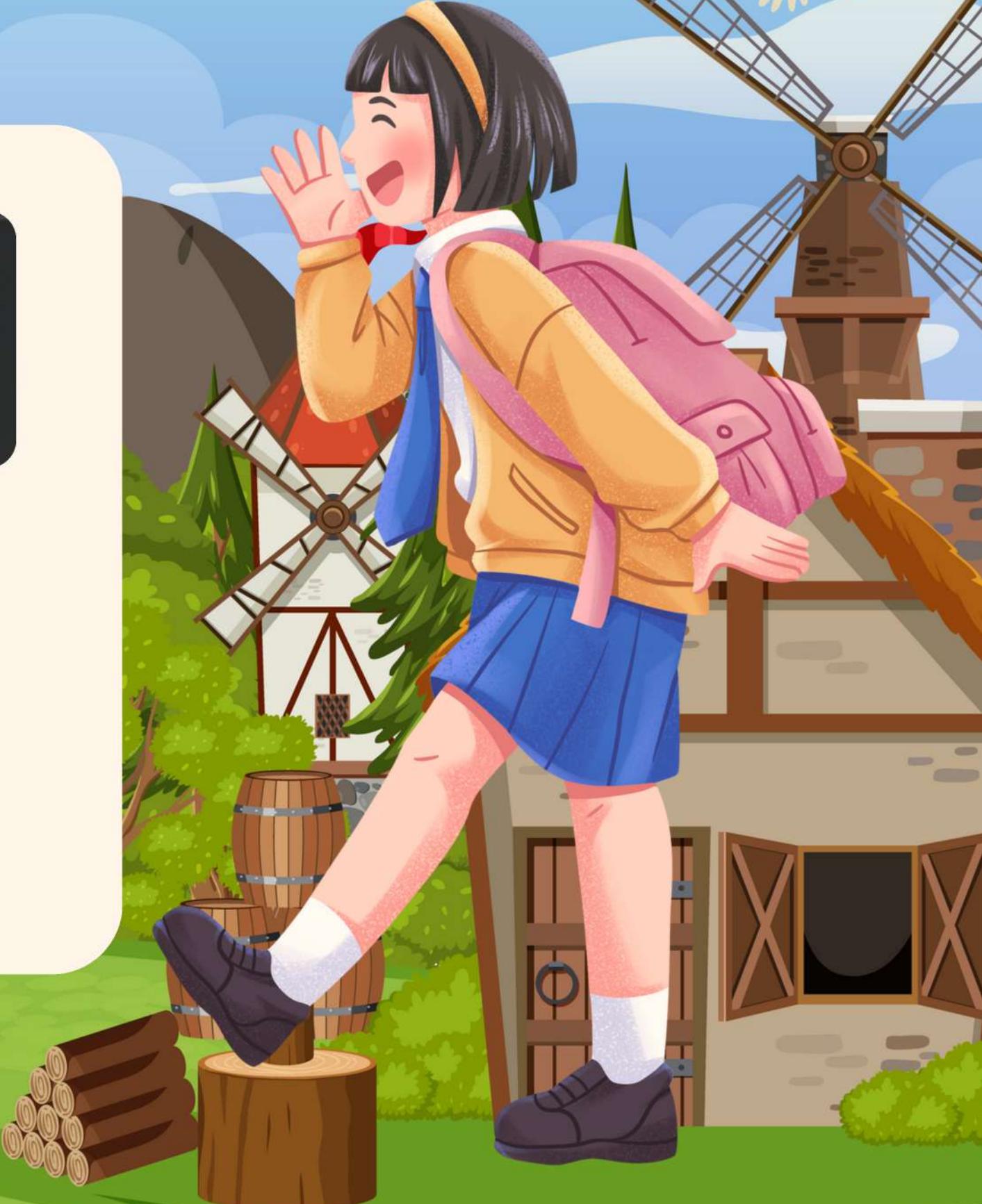
Kode ini dipakai buat mengimpor dua library, yaitu pandas untuk ngolah data tabel dan numpy untuk ngolah data angka supaya proses analisis jadi lebih mudah dan rapi.



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

### Penjelasan

“Kode ini digunakan untuk membuka fitur upload di Google Colab supaya kita bisa memasukkan file dari laptop ke lingkungan kerja Python.”



```
import pandas as pd

movies_df = pd.read_csv('movies.csv',
                       usecols=['movieId', 'title'], #usecols itu column
                       dtype={'movieId': 'int32', 'title': 'str'}) #dtype itu tipe datanya
```

### Penjelasan

Kode ini membaca file movies.csv lalu hanya mengambil kolom movieId dan title sambil memastikan tipe datanya sesuai, sehingga data film masuk ke pandas dengan rapi.

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

### Penjelasan

“Kode ini menjalankan fitur upload di Google Colab agar kita bisa memilih dan memasukkan file dari laptop ke dalam notebook.”





```
▶ ratings_df = pd.read_csv('ratings.csv',
    usecols=['userId', 'movieId', 'rating'],
    dtype={'userId': 'int32', 'movieId': 'int32', 'rating': 'float32'})
```

## Penjelasan

**“Kode ini membaca file ratings.csv dengan hanya mengambil kolom userId, movieId, dan rating sambil mengatur tipe datanya, supaya data penilaian film masuk ke pandas dengan rapi dan efisien.”**



```
movies_df.head()
```

#### Penjelasan

Kode ini menampilkan beberapa baris pertama dari movies\_df supaya kita bisa cepat ngecek isi data filmnya.

```
movies_df.shape
```

#### Penjelasan

Kode ini digunakan untuk melihat jumlah baris dan kolom di movies\_df agar kita tahu ukuran datanya.



```
ratings_df.head()
```

### Penjelasan

Kode ini digunakan untuk melihat jumlah baris dan kolom di movies\_df agar kita tahu ukuran datanya.

```
ratings_df.shape
```

### Penjelasan

Kode ini dipakai untuk melihat jumlah baris dan kolom di ratings\_df supaya kita tahu seberapa besar dataset ratingnya.

```
df = pd.merge(ratings_df, movies_df, on ='movieId')  
df
```

### Penjelasan

"Kode ini menggabungkan data rating dan data film berdasarkan movieId supaya informasi rating dan judul film jadi satu tabel."



```
▶ combine_movie_rating = df.dropna(axis=0, subset=['title'])

movie_ratingCount = (
    combine_movie_rating
        .groupby(by=['title'])['rating']
        .count()
        .reset_index()
        .rename(columns={'rating': 'totalRatingCount'})
    [['title', 'totalRatingCount']]
)

movie_ratingCount
```

### Penjelasan

Menyaring data yang judulnya lengkap, lalu ngelompokkan berdasarkan title buat ngitung berapa banyak rating tiap film, dan hasilnya jadi tabel berisi judul serta jumlah total ratingnya.



```
▶ rating_with_totalRatingCount = combine_movie_rating.merge(movie_ratingCount, left_on = 'title', right_on = 'title', how = 'left')
rating_with_totalRatingCount.head()
```

### Penjelasan

Menggabungkan data rating asli dengan tabel jumlah rating per judul supaya tiap film langsung keliatan total ratingnya dalam satu dataframe.



```
▶ pd.set_option('display.float_format', lambda x: '%.3f' % x)
print(movie_ratingCount['totalRatingCount'].describe())
```

### Penjelasan

Mengatur tampilan angka menjadi tiga desimal, lalu menampilkan ringkasan statistik dari totalRatingCount untuk memberi gambaran umum tentang sebaran jumlah rating per film.



```
popularity_threshold = 27 #batas minimal popularitas.  
rating_popular_movie= rating_with_totalRatingCount.query('totalRatingCount >= @popularity_threshold')  
rating_popular_movie.head()
```

### Penjelasan

Kami memakai threshold 27 supaya film yang kami proses memiliki cukup banyak rating untuk dianggap stabil. Angka ini tetap menjaga kualitas data, tapi tidak terlalu tinggi sehingga film non-mainstream masih bisa ikut masuk.

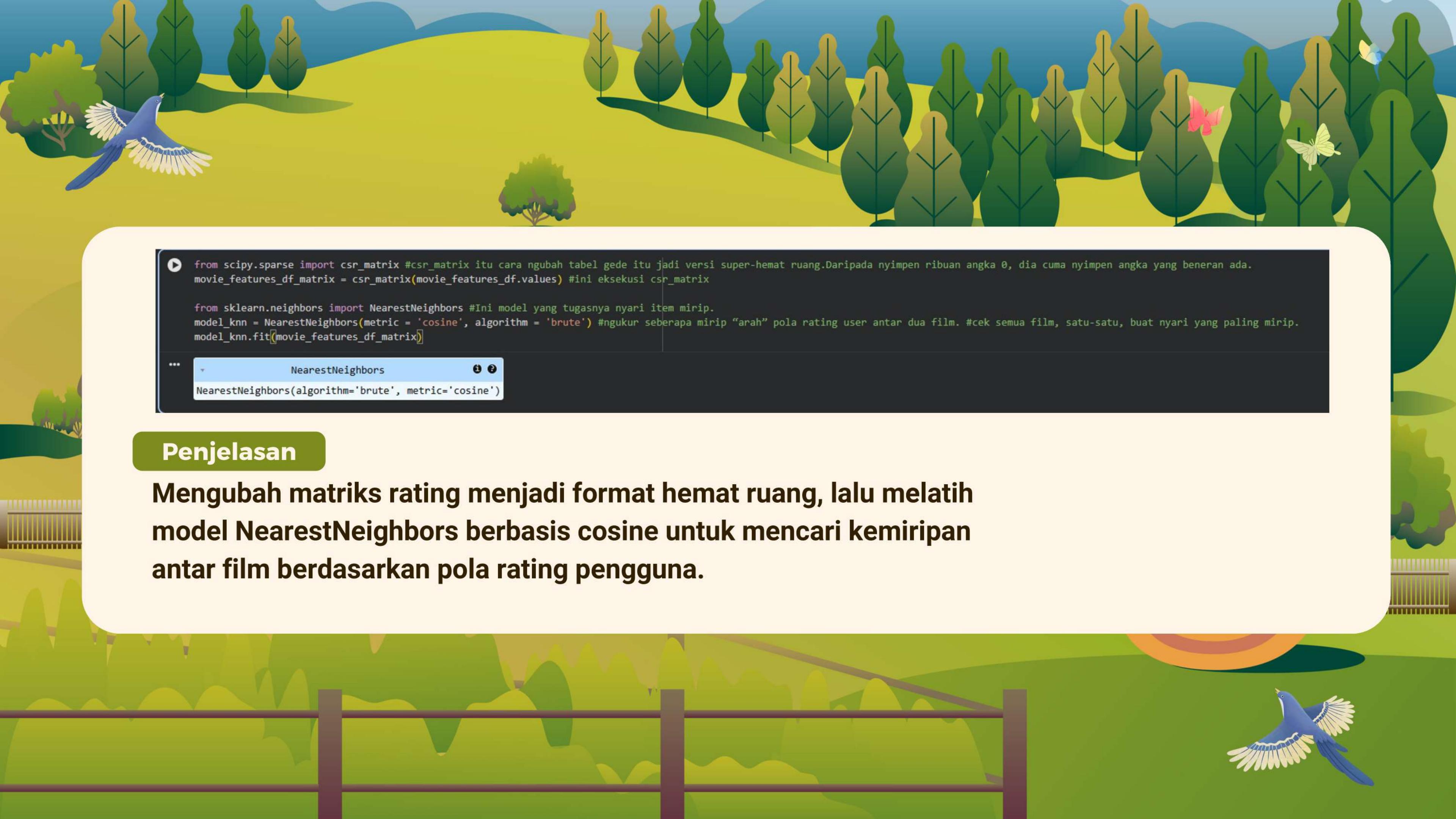


```
# First Lets create a Pivot matrix

movie_features_df=rating_popular_movie.pivot_table(index='title',columns='userId',values='rating').fillna(0)
movie_features_df.head()
```

### Penjelasan

Membentuk matriks pivot yang menyusun film sebagai baris, pengguna sebagai kolom, dan rating sebagai nilai, lalu mengisi kekosongan dengan nol agar datanya siap diproses lebih lanjut.



```
▶ from scipy.sparse import csr_matrix #csr_matrix itu cara ngubah tabel gede itu jadi versi super-hemat ruang. Daripada nyimpen ribuan angka 0, dia cuma nyimpen angka yang beneran ada.  
movie_features_df_matrix = csr_matrix(movie_features_df.values) #ini eksekusi csr_matrix  
  
from sklearn.neighbors import NearestNeighbors #Ini model yang tugasnya nyari item mirip.  
model_knn = NearestNeighbors(metric = 'cosine', algorithm = 'brute') #ngukur seberapa mirip "arah" pola rating user antar dua film. #cek semua film, satu-satu, buat nyari yang paling mirip.  
model_knn.fit(movie_features_df_matrix)  
  
...  
NearestNeighbors  
NearestNeighbors(algorithm='brute', metric='cosine')
```

## Penjelasan

Mengubah matriks rating menjadi format hemat ruang, lalu melatih model **NearestNeighbors** berbasis cosine untuk mencari kemiripan antar film berdasarkan pola rating pengguna.

movie\_features\_df.head()

...	userId	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	605	606	607	608	609	610
		title																				
10 Things I Hate About You (1999)		0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.000	0.000	3.000	0.000	5.000	0.000	0.000	0.000	0.000	
12 Angry Men (1957)		0.000	0.000	0.000	5.000	0.000	0.000	0.000	0.000	0.000	0.000	...	5.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	
2001: A Space Odyssey (1968)		0.000	0.000	0.000	0.000	0.000	0.000	4.000	0.000	0.000	0.000	...	0.000	0.000	5.000	0.000	5.000	0.000	3.000	0.000	4.500	
28 Days Later (2002)		0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.000	0.000	0.000	0.000	0.000	0.000	0.000	3.500	0.000	
300 (2007)		0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	3.000	...	0.000	0.000	0.000	3.000	0.000	0.000	5.000	0.000	4.000	

## Penjelasan

Menampilkan lima baris pertama dari movie\_features\_df untuk memberikan gambaran awal tentang susunan matriks film-user setelah dipivot.

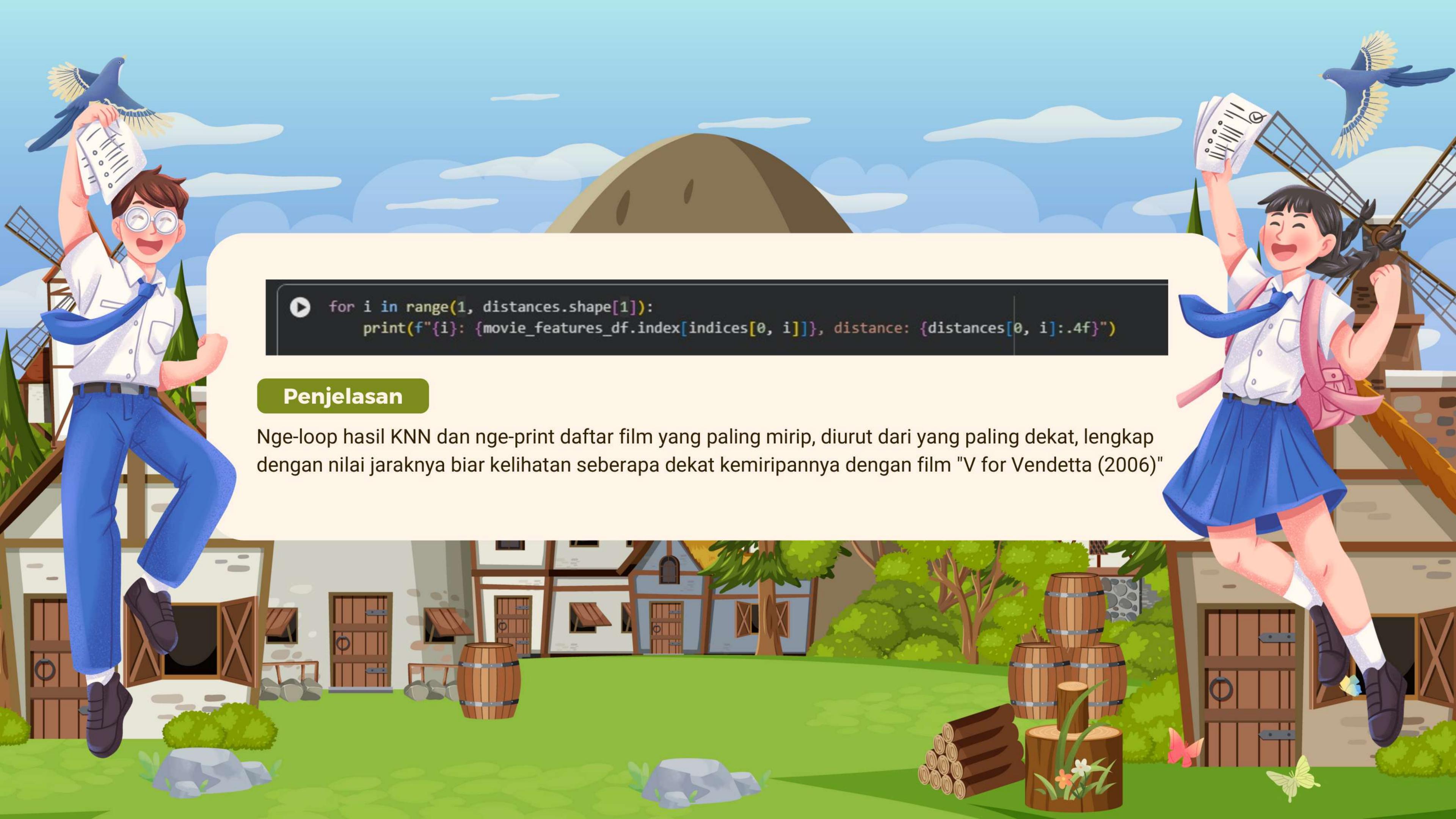


```
query_index = np.random.choice(movie_features_df.shape[0])
print(query_index)

distances, indices = model_knn.kneighbors(movie_features_df.iloc[query_index,:].values.reshape(1, -1), n_neighbors = 10)
```

### Penjelasan

Memilih satu film secara acak sebagai titik acuan, lalu menggunakan model KNN untuk mencari 10 film yang paling mirip beserta jaraknya berdasarkan pola rating pengguna.



```
for i in range(1, distances.shape[1]):  
    print(f"{i}: {movie_features_df.index[indices[0, i]]}, distance: {distances[0, i]:.4f}")
```

### Penjelasan

Nge-loop hasil KNN dan nge-print daftar film yang paling mirip, diurut dari yang paling dekat, lengkap dengan nilai jaraknya biar kelihatan seberapa dekat kemiripannya dengan film "V for Vendetta (2006)"



```
▶ movie_name = "V for Vendetta (2006)" # film yang mau dijadikan acuan  
query_index = movie_features_df.index.get_loc(movie_name)  
  
distances, indices = model_knn.kneighbors(  
    movie_features_df.iloc[query_index, :].values.reshape(1, -1),  
    n_neighbors=27  
)
```

### Penjelasan

**(Input Judul)** Menentukan sendiri judul film yang ingin dijadikan acuan analisis, jadi sistem tidak memilih film secara acak (random).

```
▶ for i in range(0, len(distances.flatten())):  
    if i == 0:  
        print('Recommendations for {0}:\n'.format(movie_features_df.index[query_index]))  
    else:  
        print('{0}: {1}, with distance of {2}:'.format(i, movie_features_df.index[indices.flatten()[i]], distances.flatten()[i]))
```

### Penjelasan

**(Cari Posisi)** Menyuruh sistem mencari lokasi data film tersebut di dalam tabel secara otomatis agar siap diproses dan memunculkan rekomendasinya



# Terima Kasih