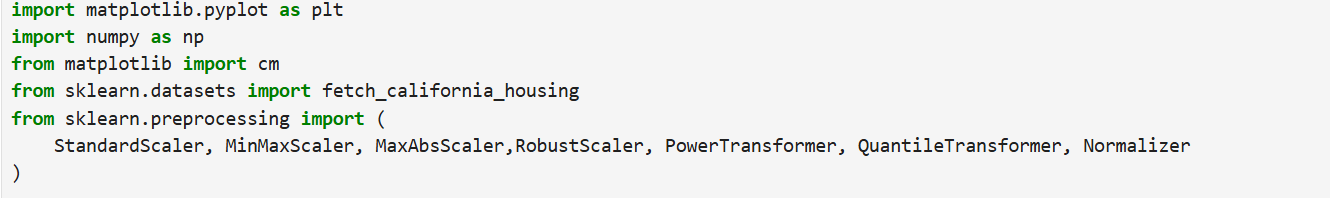
All Scalling



1.import matplotlib.pyplot as plt

Library plotting(gambar, histogram, scatter, dsb). plt dipakai untuk membuat figure, axes, memberi label, menampilkan gambar.

2.import numpy as np

Array numerik, operasi vektorial/pematriks cepat. Digunakan untuk manipulasi data sebelum plotting / modeling.

3.from matplotlib import cm

Modul colormap \_\_ memberi akses ke palet warna (mis.cm.get\_cmap(‘tab10’), cm.viridis) untuk mewarnai plot.

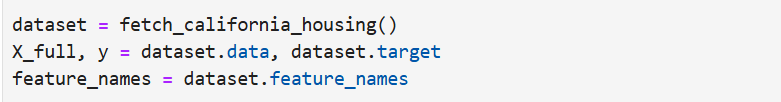
4.from sklearn.datasets import fetch\_california\_housing

Fungsi untuk memuar dataset California housing(fitur numerik seperti MedInc, HouseAge,AveRoom,dsb). Mengembalikan objek dengan data,target, feature\_names.

5.from sklearn.preprocessing import (StandardScaler, MinMaxScaler, MasxAAbsScaleer, RobustScaler, PowerTransformer, QuantileTransformer, Normalizer)

Sekumpilan transformer untuk pra-proses data numerik – tiap objek mengubah skala/distribusi fitur supaya model ML bekerja lebih baik atau lebih stabil.

2.Penjelasan detail



1.Dataset = fetch\_california\_housing()

* Fungsi : memanggil dataset bawaan dari scikit-learn, yaitu *California Housing Dataset.*
* Isi dataset : Informasi tentang rumah di California berdasarkan sensus tahun 1990.

1. Data berisi 8 fitur numerik (variabel input).
2. Target ( y ) berisi harga rumah median(dalam satuan *hundreds of thousands of dollars).*

2.X\_full, y = dataset.data, dataset.target

* Dataset.data -> X\_full
  + Bentuk array 2D (n\_samples x n\_features).
  + Berisi semua fitur (variabel independen) untuk setiap sampel rumah.
  + Misalnya : median income, house age, jumlah kamar rata – rata, dll
* Dataset.target -> y
  + Bentuk : array 1D (n\_samples).
  + Berisi **variabel target** yang ingin diprediksi : **median house value**

(harga rumah median per block).

3.Feature\_names = dataset.feature\_names

* Fungsi: menyimpan daftar nama fitur dalam bentuk list.
* Ini memudahkan kita tahu **kolom ke- berapa itu mewakili apa.**
* Daftar fitur California Housing:

|  |  |  |
| --- | --- | --- |
| Indeks | Nama Fitur | Arti |
| 1 | MedInc | Median income(pendapatan median di block, dalam puluhan ribu dolar) |
| 2 | HouseAge | Umur median rumah di block |
| 3 | AveRooms | Rata-rata jumlah kamar per rumah tangga |
| 4 | AveBedrms | Rata – rata jumlah kamar tidur per rumah tangga |
| 5 | Population | Jumlah penduduk di block |
| 6 | AveOccup | Rata – rata jumlah penghuni per rumah tangga |
| 7 | Latitude | Koordinat geografis(lintang) |
| 8 | Longitude | Koordinat geografis(bujur) |

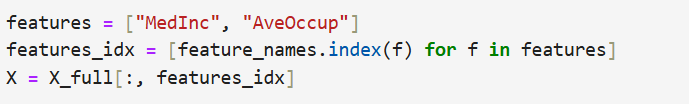
**RINGKASAN**

Dataset -> objek lengka[ hasil fetch\_california\_housing()

X\_full -> data input (fitur), shape (20640, 8).

Y -> adalah target/ output (harga rumah median).

Feature\_names-> daftar string nama fitur (agar lebih mudah membacara kolom).



1.features = [“MedInc”, “AveOccup”]

* Ini adalah **daftar nama fitur yang ingin kita pilih** dari seluruh dataset.
* Dari feature\_names (yang berisi semua 8 fitur California Housing), kita hanya ambil :
  + MedInc -> Median Income (pendapatan median per block).
  + AveOccup -> Average Occupancy (rata – rata penghuni per rumah tangga).
* Jadi, kita fokus hanya pada 2 **kolom**  dari 8 fitur.

2.features\_idx = [feature\_names.index(f) for f in features]

Feature\_name adalah list :

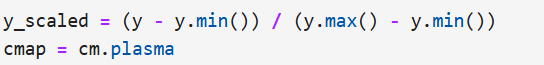
[‘MedInc’, ‘ HouseAge’, ‘AveRooms’, ‘AveBedrms’, ‘Popilation’, AveOccup’, ‘Latitude’, ‘Longitude’]

Perintah ini mencarui  **indeks posisi** dari setiap nama fitur dipilih

* “medInc” ada di posisi = 0.
* “AveOccup” ada di posisi =5.

3.X = X\_full[:, features\_idx]

* X\_full adalah array 2D (20.640 sampel x 8 fitur).
* Dengan [:, features\_idx] -> kita ambil **semua baris** ( : ) tapi hanya  **kolom dengan indeks tertentu ( [ 0, 5] ).**
* Hasil X adalah array 2D dengan shape (20640, 2) yang hanya berisi dua fitur:
  + - Kolom pertama = MedInc
    - Kolom kedua = AveOccup

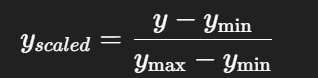


1. Y\_scalled = (y – y.min()) / (y.max() – y.min())

Ini adalah  **normalisasi min – max manual**  untuk variabel target y.

* Y -> target ( harga rumah median ).
* Y.min() -> nilai target terkecil.
* Y.max() -> nilai target terbesar.

Rumus



Hasil ada di y\_scalled akan berada di range [ 0,1 ].

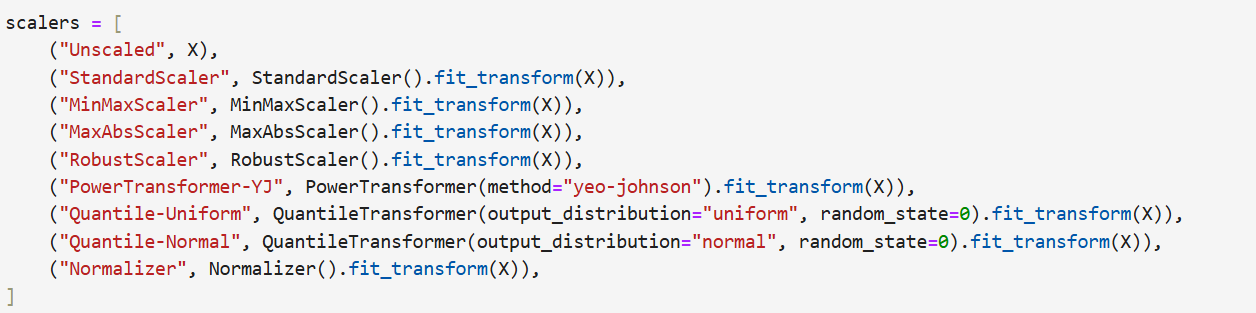
1. Cmap = cm.plasma

Cm adalah modul matplotlib.cm (colormap).

Plasma adalah salah satu  **skema warna gradien bawaan Matplotlib.**

**Warna bergerak dari gelap ungu → merah → oranye → kuning terang.**

**Contoh lain: cm.viridis, cm.inferno, cm.magma, cm.cividis.**



**1. ("Unscaled", X)**

* Data asli, **belum diproses**.
* Dipakai sebagai pembanding sebelum scaling.

**2. ("StandardScaler", StandardScaler().fit\_transform(X))**

* Standarisasi → mean = 0, standar deviasi = 1 per fitur.
* Cocok untuk model linear, SVM, PCA.

**3. ("MinMaxScaler", MinMaxScaler().fit\_transform(X))**

* Skala tiap fitur ke **rentang [0, 1]** (default).
* Umum dipakai di neural network.

**4. ("MaxAbsScaler", MaxAbsScaler().fit\_transform(X))**

* Skala tiap fitur ke [-1, 1] berdasarkan nilai absolut maksimum.
* Preservasi data sparse (banyak nol).

**5. ("RobustScaler", RobustScaler().fit\_transform(X))**

* Skala pakai **median** dan **IQR** (interquartile range).
* Tahan terhadap outlier.

**6. ("PowerTransformer-YJ", PowerTransformer(method="yeo-johnson").fit\_transform(X))**

* Transformasi non-linear → bikin distribusi fitur lebih normal (Gaussian-like).
* **Yeo-Johnson** bisa dipakai meskipun data ada nilai negatif.

**7. ("Quantile-Uniform", QuantileTransformer(output\_distribution="uniform", ...))**

* Mapping berdasarkan **quantile** → distribusi output jadi seragam [0,1].
* Mengurangi pengaruh outlier.

**8. ("Quantile-Normal", QuantileTransformer(output\_distribution="normal", ...))**

* Sama seperti di atas, tapi hasilnya distribusi normal (Gaussian).
* Bagus untuk data sangat skewed.

**9. ("Normalizer", Normalizer().fit\_transform(X))**

* Normalisasi **per baris (sample)** → tiap vektor punya norma = 1.
* Beda dengan scaler lain (yang bekerja per kolom).
* Cocok untuk teks/embedding (misalnya TF-IDF + cosine similarity).



**1. plt.subplots(...)**

Fungsi dari Matplotlib untuk membuat **figure** (kanvas utama) dan **grid of axes** (sub-plot).

* fig → objek figure, wadah besar tempat semua plot diletakkan.
* axes → array 2D berisi objek Axes individual (setiap kotak plot).

**2. len(scalers)**

* scalers adalah list berisi 9 macam scaler (Unscaled, StandardScaler, MinMax, dst).
* Jadi len(scalers) = 9.

**3. plt.subplots(len(scalers), 2, ...)**

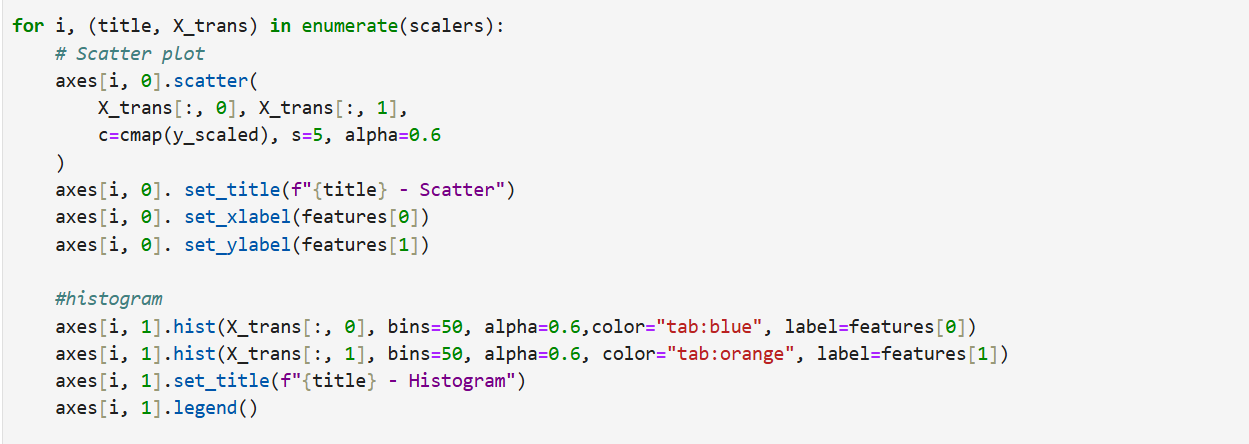
* Artinya: buat grid subplot dengan **9 baris × 2 kolom**.
* Jadi total subplot = 9 × 2 = 18.
* Masing-masing baris akan menampilkan hasil 1 scaler (contohnya histogram di kiri, scatter plot di kanan).

**4. figsize=(14, 28)**

* Mengatur ukuran kanvas figure dalam **inch** → lebar = 14, tinggi = 28.
* Karena subplot banyak (18), tingginya dibuat besar agar plot tidak berhimpitan.

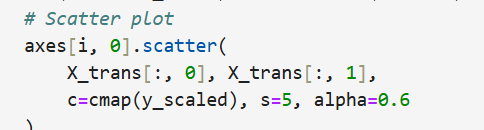
**5. Hasil variabel**

* fig → objek figure utama.
* axes → array numpy bentuk (9, 2) yang berisi semua axes.
  + Contoh:
    - axes[0, 0] → subplot baris 1 kolom 1.
    - axes[0, 1] → subplot baris 1 kolom 2.
    - axes[1, 0] → subplot baris 2 kolom 1, dst.





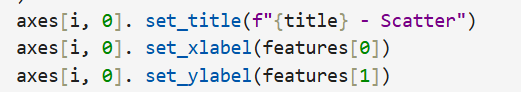
* Looping melalui list scalers.
* i = indeks scaler (0 s/d 8).
* title = nama scaler (misalnya "StandardScaler").
* X\_trans = data hasil transformasi scaler itu.



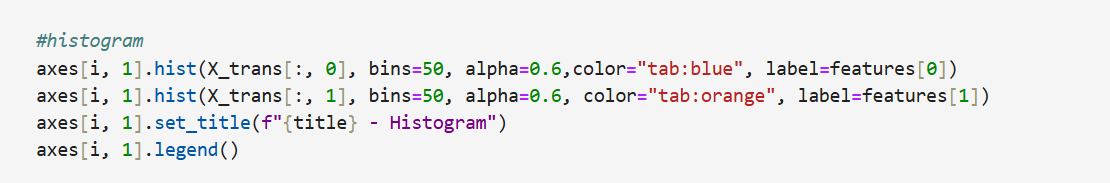
 **axes[i, 0]** → subplot kolom 0 (kiri), baris ke-i.

 scatter(...) → buat scatter plot:

* X\_trans[:, 0] → fitur pertama (mis. MedInc).
* X\_trans[:, 1] → fitur kedua (mis. AveOccup).
* c=cmap(y\_scaled) → warna titik ditentukan target harga rumah (y\_scaled) memakai colormap plasma.
* s=5 → ukuran titik kecil.
* alpha=0.6 → transparansi supaya titik yang menumpuk tetap kelihatan.



* Memberi judul subplot: misalnya "StandardScaler - Scatter".
* Label sumbu X = MedInc, sumbu Y = AveOccup.



 **axes[i, 1]** → subplot kolom 1 (kanan), baris ke-i.

 hist(...) → buat histogram distribusi nilai fitur:

* X\_trans[:, 0] → fitur pertama (MedInc), warna biru.
* X\_trans[:, 1] → fitur kedua (AveOccup), warna oranye.
* bins=50 → bagi nilai ke 50 interval.
* alpha=0.6 → transparan supaya dua histogram bisa tumpang tindih.



**1.plt.tight\_layout()**

* Fungsi untuk **merapikan tata letak subplot** di dalam figure.
* Tanpa ini, judul, label sumbu, atau legenda bisa **saling tumpang tindih** antar subplot.
* tight\_layout() otomatis mengatur margin, jarak antar subplot, dan padding supaya semua teks terlihat rapi.

**2. plt.show()**

* Perintah untuk **menampilkan figure** yang sudah dibuat.
* Kalau di Jupyter Notebook, sering otomatis tampil, tapi tetap best practice pakai plt.show() agar konsisten.
* Jika tanpa plt.show(), pada beberapa IDE (misalnya script Python biasa), grafik tidak akan muncul.