

PERTEMUAN 4

LATIHAN PRAKTIKUM FEATURE ENGINEERING

- Nama : Kinanti Anggraeni
- NIM : 4112322010

Import Dataset

```
import pandas as pd
```

```
df = pd.read_csv('/content/used_cars_dataset.csv')
df
```

	price	year	mileage	fuel_type	engine_size	transmission	num_owners	car_brand	color
0	20795	2007	141926	Petrol	1.2	Automatic	1	Tesla	Blue
1	5860	2002	100084	Hybrid	1.2	Automatic	4	Tesla	Black
2	43158	2002	179695	Petrol	1.0	Manual	4	Honda	Gray
3	49732	2000	17392	Diesel	1.0	Manual	1	Toyota	Red
4	16284	2004	65680	Petrol	1.0	Manual	1	Honda	Blue
...
95	7027	2016	87505	Electric	1.5	Automatic	1	Ford	Blue
96	7695	2016	12869	Diesel	2.5	Manual	2	Tesla	Gray
97	10258	2001	60108	Electric	3.0	Automatic	2	Tesla	Blue
98	27002	2001	122296	Electric	1.0	Manual	4	BMW	Gray
99	44504	2021	104179	Diesel	2.5	Manual	1	Tesla	Black

Langkah berikutnya:

[Buat kode dengan df](#)
[Lihat plot yang direkomendasikan](#)
[New interactive sheet](#)

Cek missing values

```
# Cek jumlah missing values di setiap kolom
missing_values = df.isnull().sum()
```

```
# Menampilkan kolom yang memiliki missing values beserta jumlahnya
missing_values[missing_values > 0]
```

```
0
```

tidak terdapat missing value pada dataset ini sehingga tidak memerlukan imputasi

Ubah fitur kategorikal menjadi numerik

```
categorical_features = ['fuel_type', 'transmission', 'car_brand', 'color']
```

```
# Menampilkan kategori unik di setiap fitur
for col in categorical_features:
    print(f"{col}:")
    print(df[col].unique()) # Menampilkan kategori unik
    print("="*50)
```

```
fuel_type:
['Petrol' 'Hybrid' 'Diesel' 'Electric']
=====
transmission:
['Automatic' 'Manual']
=====
car_brand:
['Tesla' 'Honda' 'Toyota' 'BMW' 'Ford']
=====
color:
['Blue' 'Black' 'Gray' 'Red' 'White']
=====
```

melihat masing-masing kategori unik setiap fitur

```
# Melakukan One-Hot Encoding
df_baru = pd.get_dummies(df, columns=['fuel_type', 'transmission', 'car_brand', 'color'], drop_first=True)

# Konversi nilai boolean ke integer (0 dan 1)
# nilai False setara dengan 0 dan True setara dengan 1.
df_baru = df_baru.astype(int)

# Menampilkan hasil One-Hot Encoding
print("\nData Setelah One-Hot Encoding:")
df_baru.head()
```



Data Setelah One-Hot Encoding:

	price	year	mileage	engine_size	num_owners	fuel_type_Electric	fuel_type_Hybrid	fuel_type_Petrol	transmission_Manual	car_brand_Ford	car_brand_
0	20795	2007	141926	1	1	0	0	1	0	0	
1	5860	2002	100084	1	4	0	1	0	0	0	
2	43158	2002	179695	1	4	0	0	1	1	0	
3	49732	2000	17392	1	1	0	0	0	1	0	
4	16284	2004	65680	1	1	0	0	1	1	0	

Langkah berikutnya: [Buat kode dengan df_baru](#) [Lihat plot yang direkomendasikan](#) [New interactive sheet](#)

- Dampak dari One-Hot Encoding terhadap jumlah fitur dalam dataset

Dampak dari One-Hot Encoding adalah penambahan jumlah fitur dalam dataset. Setiap kategori unik dalam fitur yang di-encode akan menjadi kolom baru. Misalnya, fitur fuel_type dengan 4 kategori (Petrol, Hybrid, Diesel, Electric), setelah One-Hot Encoding, akan mendapatkan 3 kolom baru (karena menggunakan drop_first=True untuk menghindari multikolinearitas).

Jika memiliki fitur dengan n kategori, maka One-Hot Encoding akan menambah n-1 kolom baru ke dataset.

- Efektivitas One-Hot Encoding jika terdapat banyak kategori unik

Jika dataset memiliki 100 kategori unik dalam satu fitur, One-Hot Encoding mungkin tidak efektif karena akan menghasilkan 99 kolom baru. Ini dapat menyebabkan Curse of Dimensionality (Dengan banyaknya fitur, model dapat menjadi lebih kompleks dan sulit untuk dilatih) dan juga Overfitting (Model mungkin belajar noise dalam data daripada pola yang sebenarnya)

✓ scaling pada Fitur Numerik

```
from sklearn.preprocessing import StandardScaler

# Memilih fitur numerik untuk scaling
features_to_scale = ['mileage', 'engine_size', 'num_owners']

# Inisialisasi StandardScaler
scaler = StandardScaler()

# Melakukan scaling
df_baru[features_to_scale] = scaler.fit_transform(df[features_to_scale])

# Menampilkan hasil setelah scaling
print("\nData Setelah Scaling dengan StandardScaler:")
print(df_baru[features_to_scale].head())
```



Data Setelah Scaling dengan StandardScaler:

	mileage	engine_size	num_owners
0	0.697751	-0.948936	-1.189562
1	-0.078852	-0.948936	1.306023
2	1.398758	-1.225191	1.306023
3	-1.613648	-1.225191	-1.189562
4	-0.717404	-1.225191	-1.189562

- Perbedaan Hasil antara StandardScaler, MinMaxScaler, dan RobustScaler

StandardScaler:

Mengubah data sehingga memiliki rata-rata 0 dan deviasi standar 1. Cocok untuk data yang terdistribusi normal. Sensitif terhadap outlier.

MinMaxScaler:

Mengubah data ke dalam rentang [0, 1] (atau rentang lain yang ditentukan). Cocok untuk data yang tidak terdistribusi normal dan ketika kita ingin mempertahankan proporsi data. Tidak sensitif terhadap outlier, tetapi outlier dapat mempengaruhi rentang.

RobustScaler:

Menggunakan median dan interquartile range (IQR) untuk scaling. Cocok untuk data dengan banyak outlier. Mengurangi pengaruh outlier pada scaling.

- Dalam Kondisi Apa Lebih Baik Menggunakan MinMaxScaler dibandingkan StandardScaler?

lebih baik menggunakan MinMaxScaler dalam kondisi:

Data Tidak Terdistribusi Normal: Jika data tidak mengikuti distribusi normal, MinMaxScaler dapat memberikan hasil yang lebih baik.

Model yang Sensitif terhadap Skala: Beberapa algoritma, seperti K-Nearest Neighbors (KNN) dan Neural Networks, dapat bekerja lebih baik dengan data yang terletak dalam rentang yang sama.

Preservasi Proporsi Data: Jika kita ingin mempertahankan proporsi data asli dan tidak ingin outlier mempengaruhi skala, MinMaxScaler adalah pilihan yang baik.

✓ Buat fitur baru car_age

```
# Menentukan tahun saat ini
current_year = 2025

# Membuat fitur baru car_age
df_baru['car_age'] = current_year - df['year']

# Menampilkan hasil setelah menambahkan fitur car_age
print("\nData Setelah Menambahkan Fitur car_age:")
print(df_baru[['year', 'car_age']].head())
```



```
Data Setelah Menambahkan Fitur car_age:
   year  car_age
0  2007        18
1  2002        23
2  2002        23
3  2000        25
4  2004        21
```

- Apakah fitur car_age lebih informatif dibandingkan fitur asli year?

fitur car_age lebih informatif dibandingkan year dalam analisis harga mobil bekas karena lebih langsung merepresentasikan usia mobil, yang berpengaruh terhadap nilai jual dan biaya perawatan. Selain itu, car_age menyederhanakan analisis dan mengurangi noise dengan memberikan konteks lebih jelas tentang lama pemakaian mobil.

Namun, fitur year tetap berguna untuk mengetahui tahun produksi mobil.

✓ fitur baru mileage_per_year

```
# Membuat fitur baru mileage_per_year
df_baru['mileage_per_year'] = df_baru['mileage'] / df_baru['car_age']

# Menampilkan hasil setelah menambahkan fitur mileage_per_year
print("\nData Setelah Menambahkan Fitur mileage_per_year:")
print(df_baru[['mileage', 'car_age', 'mileage_per_year']].head())
```



```
Data Setelah Menambahkan Fitur mileage_per_year:
   mileage  car_age  mileage_per_year
0  0.697751     18      0.038764
1 -0.078852     23     -0.003428
2  1.398758     23      0.060816
3 -1.613648     25     -0.064546
4 -0.717404     21     -0.034162
```

- Mengapa mileage_per_year Bisa Menjadi Fitur yang Lebih Informatif Dibandingkan mileage Saja?

fitur mileage_per_year lebih informatif dibandingkan mileage saja karena memberikan konteks tentang seberapa intens mobil digunakan setiap tahunnya. Hal ini memudahkan perbandingan antar mobil dengan usia berbeda, menjadi indikator kondisi mobil, serta lebih relevan dalam penilaian harga mobil bekas. Dengan mileage_per_year, kita dapat memahami pola penggunaan mobil secara lebih akurat dibandingkan hanya melihat total jarak tempuhnya.

✓ Menampilkan dataset terbaru

```
print("\nDataset Terbaru Setelah Menambahkan Fitur Baru:")
df_baru.head()
```



Output:

owners	fuel_type_Electric	fuel_type_Hybrid	fuel_type_Petrol	transmission_Manual	car_brand_Ford	car_brand
39562	0	0	1	0	0	
36023	0	1	0	0	0	
36023	0	0	1	1	0	
39562	0	0	0	1	0	
39562	0	0	1	1	0	

Langkah berikutnya: [Buat kode dengan df_baru](#) [Lihat plot yang direkomendasikan](#) [New interactive sheet](#)

Mulai coding atau [buat](#) kode dengan AI.