DASPRO

Enterprise Data Management Laboratory

Fakultas Rekayasa Industri
Telkom University

Data Prepocessing

Modul Praktikum
Penambangan Data 2025

MODUL 1

Arranged By
DASPRO Laboratory Team

Asisten Praktikum

AANG
Arya Anggadipa Manova

AAAA
Alvaro Cleosanda

JHON
I Gede Made Ari Ananta

KAYE
Kaisa Izzatunniswah

KYLE
Kayla Zhafira Ardinov

KYRA
Kirana Adira Syifa

LALA
Lailatul Hadhari

MALE
Matthew Alexander H Sitorus

MIIZ
Mochamad Irdan Iqbal Z

NICE
Nisa Trinanda Utami

PDLY
Muhammad Fadli M

QIRA
Ariq Naufal Wahyono

QLAB
Balqis Eka Nurfadisyah

RHAN
Sarah Luki Raihani

Asisten Praktikum

RIAU
Alfan Gunawan Akhmad

RIDH
Muhammad Ridho Irhamna

SATH
Muhammad Salik Arethe

SRSA
Shalvia Retno Salsabil

TSAA
Maitsa Luthfiyyah Durrotunnisa

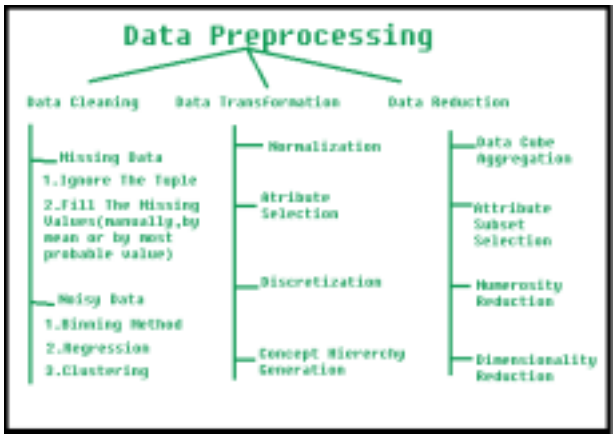
WANG
Ikhwan Amiruddin

XENA
Goesniawan Xena Adikara

ZURU
Daffa Muhammad Zulfikar

Data Preprocessing

Data preprocessing adalah tahapan awal yang sangat penting dalam *data mining*, karena kualitas data yang digunakan akan sangat mempengaruhi hasil analisis atau prediksi yang dibuat. Hal ini mengacu pada pembersihan, transformasi, dan integrasi data agar siap untuk dianalisis. Data mentah yang diambil dari sumber asli seringkali tidak siap digunakan secara langsung karena adanya masalah seperti data yang hilang, *noise*, atau *inconsistency*. Oleh karena itu, *preprocessing* bertujuan untuk membersihkan dan mengubah data agar lebih sesuai untuk analisis lebih lanjut. Langkah-langkah spesifik dalam *data preprocessing* dapat bervariasi, tergantung pada sifat data dan tujuan analisis.



Langkah-langkah umum dalam *data preprocessing.*
Sumber: www.geeksforgeeks.org/data-preprocessing-in-data-mining

Di Python, terdapat banyak *libraries* yang tersedia untuk melakukan *data preprocessing.* Pemilihan *libraries* yang digunakan dapat disesuaikan dengan kebutuhan dari *dataset* terkait. Berikut beberapa contoh *library* yang umum digunakan untuk *data preprocessing.*

Tahapan pada Data *Preprocessing*	**Library**	***Syntax* relevan**
Data understanding: Loading data	pandas	• pd.read_csv(): membaca data dari csv • pd.read_excel(): membaca data dari excel • pd.read_sql_query(): membaca data dari sql
Data understanding: menampilkan karakteristik data	pandas	• head(): Menampilkan beberapa baris pertama data. • tail(): Menampilkan beberapa baris terakhir data. • head(n): Menampilkan n baris pertama. Defaultnya menampilkan 5 baris. • tail(n): Menampilkan n baris terakhir. Defaultnya menampilkan 5 baris. • describe(): Menyediakan statistik deskriptif seperti mean, std, min, max, dll. • info(): Menampilkan informasi tentang DataFrame, termasuk tipe data dan jumlah nilai kosong. • value_counts(): Menghitung frekuensi nilai unik dalam suatu kolom. • corr(): Menghitung korelasi antar variabel numerik.

		• shape: Menampilkan dimensi dari DataFrame (baris dan kolom). • dtypes: Menampilkan tipe data dari masing-masing kolom. • unique(): Menampilkan nilai unik dari suatu kolom. • nunique(): Menghitung jumlah nilai unik dari suatu kolom. • nunique(dropna=True): Menghitung jumlah nilai unik. Jika dropna=False, NaN juga dihitung. • df['column_name'].mean(): Menampilkan rata-rata • df['column_name'].median(): Menampilkan median • df['column_name'].mode(): Menampilkan modus
	numpy	• np.mean(): Menghitung rata-rata.np.median(): Menghitung median. • np.std(): Menghitung deviasi standar. • np.var(): Menghitung varians. • np.min(): Menghitung nilai minimum. • np.max(): Menghitung nilai maksimum. Dengan parameter yang bisa digunakan: – `a: array input.` – `axis: (optional) menentukan axis yang ingin dihitung.` – `dtype: (optional) tipe data hasil.`
Data understanding: visualisasi data	matplotlib	• plt.hist(data, bins, color, alpha): membuat histogram • plt.boxplot(data): membuat *boxplot* • plt.plot(x, y, color, marker): membuat *line chart* • plt.scatter(x, y, color, alpha): membuat *scatter plot* • plt.bar(x, height, color): membuat barplot • plt.pie(sizes, labels, startangle): membuat *pie chart* • Plt.xlabel: memberikan label untuk sumbu X • plt.ylabel: memberikan label untuk sumbu Y • plt.title: memberikan judul untuk graph Dengan parameter yang dapat digunakan: - data: Dataset yang akan divisualisasikan. - bins: Jumlah bin untuk histogram. - color: Warna histogram. - alpha: Transparansi (0-1) dari warna. - marker: Tipe marker untuk titik data. - height: Tinggi setiap bar (nilai frekuensi). - labels: Nama kategori untuk setiap irisan. - startangle: Sudut awal untuk memulai irisan pie. - x: Kolom untuk sumbu x. - y: Kolom untuk sumbu y.
	seaborn	• sns.histplot(data, bins=10, color='blue', kde=False) • sns.boxplot(x, y, data) • sns.lineplot(x, y, data) • sns.scatterplot(x, y, data) • sns.barplot(x, y, data) • plt.pie(sizes, labels, autopct, startangle) Dengan parameter yang dapat digunakan: - *data: Dataset yang akan divisualisasikan.* - bins: Jumlah bin untuk histogram. - color: Warna histogram.

		- kde: Jika True, tambahkan kurva kepadatan (kernel density estimate). - x: Kolom untuk sumbu x. - y: Kolom untuk sumbu y. - data: Dataset yang akan divisualisasikan.
Data cleansing: Identifikasi dan penanganan data duplikat	pandas	• duplicated(): Mengidentifikasi baris duplikat. • drop_duplicates(): Menghapus baris duplikat. • duplicated(subset, keep): Mengidentifikasi baris yang duplikat. drop_duplicates(subset=None, keep='first', inplace=False): Menghapus baris yang duplikat. Dengan parameter yang bisa digunakan: – `subset: subset menentukan kolom yang dicek` – `keep: menentukan apakah mempertahankan baris pertama ('first') atau terakhir ('last') yang muncul.` – `inplace: inplace=True untuk memodifikasi DataFrame secara langsung tanpa membuat salinan baru.`
	numpy	• np.unique(): Mengembalikan array yang berisi elemen unik dari array input, juga dapat menghitung frekuensi. Dengan parameter yang bisa digunakan: – `ar: array input.` – `return_index: (optional) jika True, mengembalikan indeks dari elemen unik.` – `return_inverse: (optional) jika True, mengembalikan indeks elemen yang membentuk array input dari array unik.` – `return_counts: (optional) jika True, mengembalikan frekuensi dari elemen unik.`
Data cleansing: Penanganan data inkonsisten	pandas	• str.upper(), str.lower(), str.title(): Mengubah format string untuk konsistensi. • replace(): Mengganti nilai tertentu dalam DataFrame dengan nilai lain. • replace(to_replace): Mengganti dengan nilai pilihan, dapat berupa scalar, list, dictionary, atau regex. • replace(value): Mendefinisikan nilai pengganti • replace(method): (Hanya untuk penggantian NaN) Menentukan cara pengisian (bisa 'pad', 'backfill', dll.). • apply(): menerapkan fungsi ke setiap elemen kolom atau DataFrame • apply(func): fungsi yang akan diterapkan • fillna(): mengisi fungsi yang hilang • fillna(value): nilai pengganti • fillna(method): metode pengisian • fillna(limit): jumlah nilai yang akan diisi • map(): mengganti nilai dengan mapping yang ditentukan. • str.replace(): mengganti bagian dari string dengan string lain. • strip(): menghapus karakter di awal dan akhir string
	Numpy	• np.where(): Mengembalikan elemen dari dua array berdasarkan kondisi. • np.char.replace(): Mengganti substring dalam array string. Dengan parameter yang bisa digunakan: - condition: kondisi untuk menentukan elemen yang diambil..

		- x: elemen yang diambil jika kondisi True. - y: elemen yang diambil jika kondisi False
Data cleansing: Mengoreksi nilai yang tidak valid	Pandas	• isna() atau isnull(): Mengidentifikasi nilai NaN. • fillna(): Mengisi nilai yang kosong. • dropna(): Menghapus baris dengan nilai kosong
	Numpy	• np.nan: Menyimpan nilai NaN (Not a Number). • np.nan_to_num(): Mengubah NaN menjadi nilai tertentu. • np.isnan(): Mengidentifikasi elemen NaN dalam array.
Data cleansing: Menangani outlier	Numpy	• np.mean(): Menghitung rata-rata. • np.std(): Menghitung deviasi standar. • np.percentile(): Menghitung nilai persentil. Dengan parameter yang bisa digunakan: – `a: array input.` – `q: persentil (antara 0–100) untuk np.percentile().`
Data reduction	Scikit-learn	• PCA(n_components): Reduksi dimensi dengan Principal Component Analysis Dengan parameter yang dapat digunakan: - n_components: Jumlah komponen utama yang diinginkan. Jika None, semua komponen akan dihitung.
	Pandas	• df.sample(n, frac, replace, random_state) Dengan parameter yang dapat digunakan: - n: Jumlah baris yang diambil sebagai sampel. - frac: Persentase sampel dari DataFrame. - replace: Jika True, sampel diambil dengan penggantian. - random_state: Seed untuk pengambilan sampel acak agar hasil konsisten.
Menangani *missing value*	Pandas	• df.notnull(): Mengembalikan DataFrame atau Series boolean yang menunjukkan nilai yang tidak kosong. • df.isnull(): Mengembalikan DataFrame atau Series boolean yang menunjukkan nilai yang kosong (missing). • df.dropna(axis, how, thresh, subset, inplace) • df.fillna(value, method, axis, inplace, limit) Dengan parameter yang dapat digunakan: - axis: 0 untuk menghapus baris, 1 untuk menghapus kolom. - how: Jika any, menghapus baris/kolom dengan setidaknya satu nilai kosong. Jika all, menghapus baris/kolom dengan semua nilai kosong. - thresh: Jumlah minimum nilai non-null yang diperlukan. - subset: Daftar kolom yang dipertimbangkan. - inplace: Jika True, perubahan dilakukan langsung pada DataFrame. - value: Nilai untuk mengisi nilai kosong, bisa berupa scalar, dictionary, atau DataFrame. - method: Metode pengisian, seperti ffill (forward fill) atau bfill (backward fill). - limit: Jumlah maksimum nilai untuk mengisi.

		• sns.heatmap(data, cmap, cbar, yticklabels): Membuat *heatmap* untuk melihat distribusi nilai kosong dalam dataset. Dengan parameter yang dapat digunakan: - data: Data yang akan divisualisasikan. - cmap: Skema warna untuk *heatmap*. - yticklabels: Menampilkan label sumbu y jika True.
Menangani *missing value*	Scikit-learn	• KNNImputer(n_neighbors, weights, metric): Mengisi nilai kosong berdasarkan K Nearest Neighbors. • imputer.fit_transform(X): Melakukan fitting dan transformasi data dalam satu langkah. Dengan parameter yang dapat digunakan: - n_neighbors: Jumlah tetangga terdekat untuk dipertimbangkan saat mengisi. - weights: Jenis bobot yang digunakan untuk nilai tetangga (uniform atau distance). - metric: Metode pengukuran jarak, default adalah nan_euclidean. - X: Dataset yang akan difit dan ditransformasikan.
Data transformation	category_encoders	• OneHotEncoder(): Mengubah kategori ke representasi biner. • BinaryEncoder(): Mengubah data kategori menjadi representasi biner. • TargetEncoder(): Melakukan encoding berdasarkan target.

A. Data Understanding

1. *Loading* data

Sebelum melakukan tahapan data *preprocessing*, sangat penting untuk memahami struktur, kualitas, dan karakteristik data yang kita miliki. Langkah pertama adalah membaca data dari sumber seperti file CSV, Excel, *database*, atau API menggunakan *syntax* berikut.

```
import pandas as pd

# Membaca data dari file CSV
df = pd.read_csv('data.csv')

# Membaca data dari file Excel
df = pd.read_excel('data.xlsx')

# Membaca data dari database SQL (contoh MariaDB atau SQLite)
import sqlite3
conn = sqlite3.connect('database.db')
df = pd.read_sql_query("SELECT * FROM table_name", conn)
```

2. *Menampilkan karakteristik data*

Setelah data berhasil dibaca, diperlukan untuk melihat karakteristik data untuk memahami bagaimana data itu disusun.

Untuk memahami dataset secara umum, kita dapat menggunakan beberapa *syntax* dasar seperti **info()** untuk melihat informasi dasar tentang tipe data dan jumlah data di setiap kolom, **head()** dan **tail()** untuk melihat beberapa baris pertama dan terakhir yang dapat membantu mengidentifikasi struktur dan format data, **describe()** untuk membantu mendapat ringkasan statistik dari data numerik, seperti rata-rata, standar deviasi, nilai maksimum, dan minimum. Berikut beberapa *syntax* yang dapat digunakan untuk eksplorasi karakteristik data setelah data berhasil dibaca dengan Pandas.

```
df.head()  # Menampilkan 5 baris pertama secara default
df.info()  # Menampilkan informasi tentang kolom dan tipe data
df.describe()  # Statistik seperti count, mean, std, min, max, dll.
df.describe(include='object')  # Statistik untuk data kategorikal/non-numerik
df.columns  # Menampilkan daftar nama kolom
df.shape  # Menghasilkan (jumlah_baris, jumlah_kolom)

df.dtypes  # Menampilkan tipe data tiap kolom
df['column_name'] = df['column_name'].astype('desired_dtype') #Mengubah tipe kolom

df['column_name'].unique()  # Menampilkan semua nilai unik pada kolom
df['column_name'].value_counts()  # Menghitung frekuensi kemunculan setiap nilai
```

Selain menggunakan **df.describe(),** untuk melihat statistik deskriptif dari dataset, Pandas juga menyediakan syntax khusus untuk menghitung statistik berdasarkan ukuran yang dipilih. Pemilihan ukuran statistik ini dapat dikategorikan berdasarkan jenis data.

a. **Data nominal/biner** terdiri dari kategori tanpa urutan, sehingga ukuran yang dapat digunakan adalah **modus.**
b. **Data ordinal** memiliki kategori dengan urutan. Misalnya, tingkat pendidikan (SD, SMP, SMA, Perguruan Tinggi). Ukuran yang dapat digunakan adalah **modus** dan **median.**
c. **Data interval dan rasio.** Data interval dan rasio memiliki nilai numerik dan dapat dihitung. Misalnya, suhu (data interval) atau berat (data rasio). Ukuran yang dapat digunakan adalah **mean, median, dan modus.**

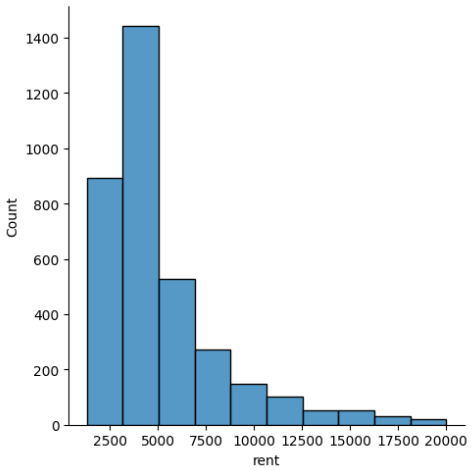
3. *Visualisasi data*

Setelah memahami data secara mendasar, langkah selanjutnya adalah memahami distribusi dan karakteristik lainnya melalui visualisasi data. Visualisasi ini sangat membantu untuk mengidentifikasi pola, persebaran, dan masalah potensial seperti *outlier*. Visualisasi yang digunakan bisa disesuaikan dengan tipe data dan tujuan analisis.

a. Variabel kuantitatif

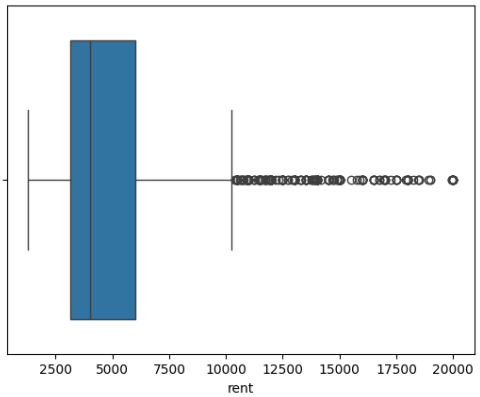
- Histogram: Menyampaikan visualisasi mengenai nilai minimum dan maksimum, lokasi pusat, penyebaran, dan juga pola yang dapat mempengaruhi analisis, seperti *skew*/kemiringan. Data dibagi menjadi interval (bin), dan tinggi batang menunjukkan jumlah data yang jatuh dalam setiap interval.

```
# Create a histogram of the rent variable
sns.displot(dataframe.rent, bins=10, kde=False)
plt.show()
```

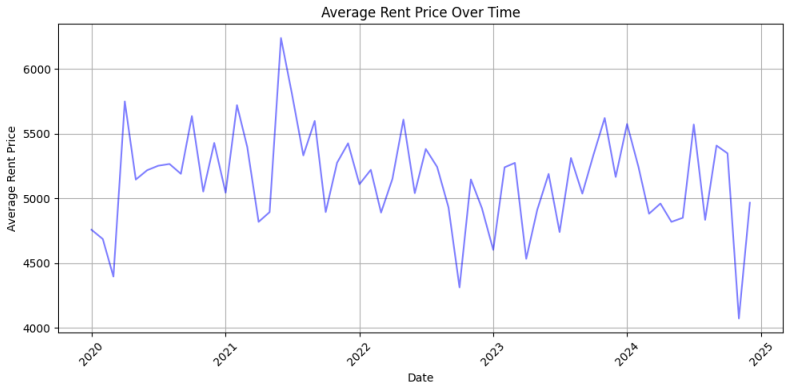


- *Boxplot*: Menyampaikan visualisasi mengenai distribusi data melalui menyajikan nilai minimum, maksimum, median, dan kuartil dari data. Titik di luar *whiskers boxplot* menunjukkan *outliers.*

```
# Load libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Create the plot
sns.boxplot(x='rent', data=dataframe)
plt.show()
```

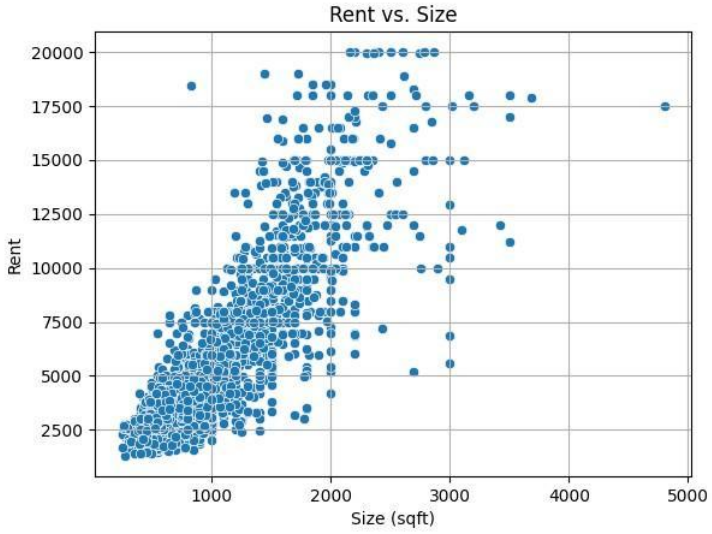


- *Line chart*: Menunjukkan hubungan antara dua variabel, di mana satu variabel biasanya adalah waktu dan yang lainnya adalah nilai numerik. Berfungsi untuk analisis tren serta perbandingan antar variabel.



- *Scatter plot*: Diagram yang menampilkan nilai dua atau lebih variabel numerik dengan titik-titik menunjukkan hubungan antara kedua variabel tersebut. Berfungsi untuk menunjukkan hubungan seperti korelasi positif, negatif, atau tidak ada. Digunakan juga untuk visualisasi titik-titik yang jauh dari pola umum, yang mungkin menjadi outlier.

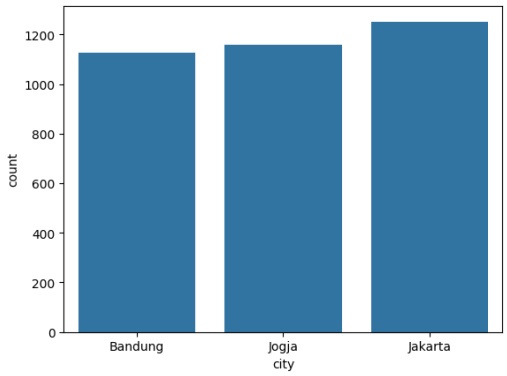
```
import seaborn as sns
import matplotlib.pyplot as plt

# Create a scatter plot of the size_sqft variable relative to rent
sns.scatterplot(x=dataframe['size_sqft'], y=dataframe['rent'])
plt.xlabel('Size (sqft)')
plt.ylabel('Rent')
plt.title('Scatter Plot of Rent vs. Size')
plt.grid()
plt.show()
```



b. Variabel kuantitatif

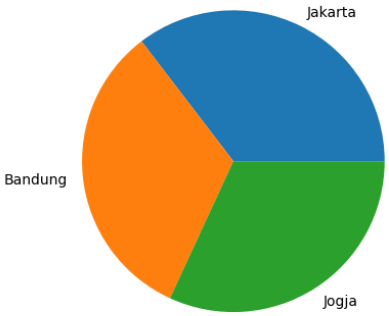
- *Bar plot*: Menampilkan dan membandingkan frekuensi atau nilai antar kategori.

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.countplot(x='city',  data=dataframe)
plt.show()
```



- *Pie chart: M*enunjukkan kontribusi relatif setiap kategori terhadap total keseluruhan

```
# Define the labels in pie chart
cities_labels = ["Jakarta", "Bandung", "Jogja"]

# Generate pie chart of cities
plt.pie(dataframe.city.value_counts(), labels = cities_labels)
plt.show()
```



B. Data Cleansing

Data cleansing, atau pembersihan data, adalah proses identifikasi dan perbaikan atau penghapusan data yang tidak akurat, rusak, tidak lengkap, duplikat, atau tidak relevan dari *dataset*. Proses ini sangat penting untuk memastikan data yang digunakan dalam analisis akurat dan dapat diandalkan. *Data cleansing* melibatkan berbagai langkah, mulai dari validasi data, koreksi kesalahan, penghapusan duplikasi, hingga pengisian data yang hilang.

1. Identifikasi dan penanganan data duplikat

Data duplikat terjadi ketika satu atau lebih baris dalam *dataset* muncul lebih dari sekali, baik secara keseluruhan atau sebagian. Hal ini bisa disebabkan oleh kesalahan penginputan, penggabungan dataset yang berbeda, atau penyalinan yang tidak disengaja. Mengabaikan duplikat dapat mengakibatkan bias, karena data yang berulang akan memberi bobot lebih besar terhadap pola tertentu yang mungkin tidak merepresentasikan keseluruhan *dataset* dengan benar.

```
import pandas as pd

# Dataset dengan duplikat
data = {'ID': [1, 2, 3, 3, 4, 5, 6, 6],
        'Name': ['A', 'B', 'C', 'D', 'E', 'F', 'F', 'F']}
df = pd.DataFrame(data)

# Menghitung duplikat
duplicates = df.duplicated().sum()
print(duplicates)
```

```
1
```

```
[5] # Identifikasi duplikat
duplicates = df[df.duplicated()]
print("Duplikat yang ditemukan:\n", duplicates)

# Menghapus duplikat berdasarkan semua kolom
df_cleaned = df.drop_duplicates()
print("\nData setelah duplikat dihapus:\n", df_cleaned)

# Jika hanya ingin menghapus duplikat berdasarkan satu kolom, misalnya 'ID'
df_cleaned_by_id = df.drop_duplicates(subset=['ID'])
print("\nData setelah menghapus duplikat berdasarkan kolom ID:\n", df_cleaned_by_id)
```

```
Duplikat yang ditemukan:
   ID Name
3  3    C
7  6    F

Data setelah duplikat dihapus:
   ID Name
0  1    A
1  2    B
2  3    C
4  4    D
5  5    E
6  6    F

Data setelah menghapus duplikat berdasarkan kolom ID:
   ID Name
0  1    A
1  2    B
2  3    C
4  4    D
5  5    E
6  6    F
```

Pendekatan di atas menggunakan library Pandas di Python untuk identifikasi data duplikat dengan beberapa metode sebagai berikut.

a. **duplicated().sum():** menghitung jumlah baris yang memiliki nilai duplikat
b. **duplicated():** mengidentifikasi baris-baris yang diduplikasi dalam sebuah *dataframe*.
c. **drop_duplicates():** fungsi ini digunakan untuk menghapus baris yang merupakan duplikat dari *dataframe*.

2. Penanganan data inkonsisten

Data tidak konsisten terjadi ketika ada variasi dalam pengisian data yang merujuk pada hal yang sama. Misalnya, jika ada nilai yang seharusnya sama tetapi ditulis dengan format yang berbeda, seperti "Yes", "yes", "YES", atau "No", "N", dan "no". Inkonsistensi ini dapat menghambat analisis atau agregasi karena komputer menganggap nilai yang berbeda sebagai entitas terpisah, meskipun secara logika artinya sama.

```
# Data dengan format tidak konsisten
data = {'Status': ['Yes', 'No', 'yes', 'No', 'Y', 'n', 'NO']}
df = pd.DataFrame(data)

# Normalisasi data untuk memastikan konsistensi
df['Status'] = df['Status'].str.lower().replace({'y': 'yes', 'n': 'no'})
print("Data setelah dikonsistenkan:\n", df)

# Alternatif: jika ingin melakukan pemetaan manual
mapping = {'yes': 'yes', 'y': 'yes', 'no': 'no', 'n': 'no'}
df['Status'] = df['Status'].replace(mapping)
print("Data setelah pemetaan manual:\n", df)
```

```
Data setelah dikonsistenkan:
  Status
0   yes
1    no
2   yes
3    no
4   yes
5    no
6    no
Data setelah pemetaan manual:
  Status
0   yes
1    no
2   yes
3    no
4   yes
5    no
6    no
```

Pendekatan di atas menggunakan library Pandas di Python dengan teknik seperti *lowercasing* (mengubah seluruh teks menjadi huruf kecil), atau standarisasi entri dengan mengganti variasi format yang berbeda dengan fungsi replace() atau map().

3. Mengoreksi nilai yang tidak valid

Invalid values adalah data yang tidak sesuai dengan ekspektasi atau aturan tertentu. Misalnya, dalam dataset demografi, usia yang bernilai negatif atau lebih dari 150 tahun akan dianggap tidak valid. *Invalid values* sering kali muncul akibat kesalahan input data atau data yang hilang yang tidak terkelola dengan baik.

```
# Dataset dengan nilai tidak valid
data = {'Name': ['Fathima', 'Jessica', 'Cindy', 'Btari'],
        'Age': [21, -5, 200, 25]}  # Nilai -5 dan 200 adalah tidak valid
df = pd.DataFrame(data)

# Perbaiki nilai yang tidak valid dengan mengganti nilai di luar batas 0-150
df['Age'] = df['Age'].apply(lambda x: x if 0 <= x <= 150 else None)
print("Data setelah nilai tidak valid dikoreksi:\n", df)
```

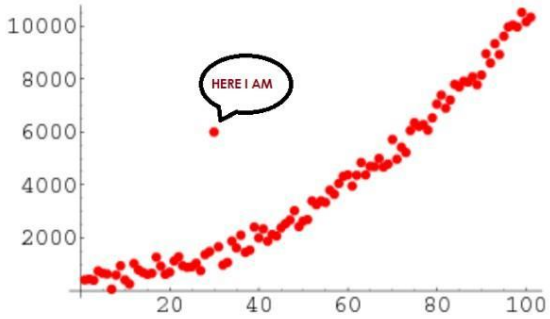
```
Data setelah nilai tidak valid dikoreksi:
     Name  Age
0  Fathima  21.0
1  Jessica   NaN
2    Cindy   NaN
3    Btari  25.0
```

Pendekatan di atas membutuhkan identifikasi nilai yang masuk akal berdasarkan logika. Setelah diidentifikasi, *invalid values* dapat diganti dengan nilai None/NaN. Nilai tidak valid ini juga dapat dihapus, atau diganti dengan nilai yang lebih sesuai menggunakan mean, median, dan sebagainya.

```
import pandas as pd
import numpy as np

# Dataset dengan nilai yang hilang (NaN) atau tidak valid
data = {'Name': ['Fathima', 'Jessica', 'Btari', 'Laras', 'Cindy'],
        'Age': [21, np.nan, 25, 35, np.nan]}  # Ada nilai NaN pada kolom 'Age'
df = pd.DataFrame(data)

print("Data sebelum pengisian nilai yang hilang:\n", df)

# Menghitung mean dari kolom 'Age', mengabaikan nilai NaN
mean_age = df['Age'].mean()

# Mengisi nilai NaN dengan mean dari kolom 'Age'
df['Age'].fillna(mean_age, inplace=True)

print("\nData setelah pengisian nilai yang hilang dengan mean:\n", df)
```

```
Data sebelum pengisian nilai yang hilang:
     Name  Age
0  Fathima  21.0
1  Jessica   NaN
2   Btari  25.0
3   Laras  35.0
4   Cindy   NaN

Data setelah pengisian nilai yang hilang dengan mean:
     Name  Age
0  Fathima  21.0
1  Jessica  27.0
2   Btari  25.0
3   Laras  35.0
4   Cindy  27.0
```

4. Noisy *data / Outlier*



Noisy data adalah data yang mengandung kesalahan atau variabilitas yang tidak dapat dijelaskan oleh pola yang diharapkan. Sumber *noise* bisa datang dari kesalahan pengukuran, pengambilan data yang buruk, atau faktor acak lainnya. Dalam dunia data, *outlier* (nilai ekstrem) juga dapat dianggap sebagai bagian dari *noisy data*. Terdapat beberapa cara dalam mengidentifikasi keberadaan *outlier* pada *dataset*, di antaranya:

a. Boxplot

   Boxplot adalah salah satu cara visual yang efektif untuk mendeteksi outliers. Outliers ditandai sebagai titik-titik di luar whisker (garis) pada *boxplot*.

```python
#Outlier identification with Boxplot
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Contoh dataset
data = {'Value': [10, 12, 15, 13, 14, 120, 130, 14, 16, 13, 12, 115, 14, 16]}
df = pd.DataFrame(data)

# Membuat boxplot
sns.boxplot(x=df['Value'])
plt.show()
```

b. Z-Score

Z-score adalah metode statistik yang menunjukkan seberapa jauh suatu nilai dari *mean* dalam satuan standar deviasi. *Outliers* umumnya dianggap sebagai nilai dengan z-score > 3 atau < -3.

```python
import pandas as pd
from scipy import stats
import numpy as np

# Contoh dataset
data = {'Value': [10, 12, 15, 13, 14, 120, 130, 14, 16, 13, 12, 115, 14, 16, 300, 5, 1]}
df = pd.DataFrame(data)

# Menghitung z-score untuk setiap nilai
df['z_score'] = stats.zscore(df['Value'])

# Menampilkan z-score setiap nilai
print("Z-scores:\n", df[['Value', 'z_score']])

# Memfilter outliers dengan z-score lebih dari 2 atau kurang dari -2
outliers = df[(df['z_score'] > 2) | (df['z_score'] < -2)]
print("\nOutliers berdasarkan Z-Score:\n", outliers)
```

```
Z-scores:
    Value   z_score
0      10 -0.506050
1      12 -0.479579
2      15 -0.439874
3      13 -0.466344
4      14 -0.453109
5     120  0.949816
6     130  1.082168
7      14 -0.453109
8      16 -0.426639
9      13 -0.466344
10     12 -0.479579
11    115  0.883641
12     14 -0.453109
13     16 -0.426639
14    300  3.332143
15      5 -0.572226
16      1 -0.625166

Outliers berdasarkan Z-Score:
    Value   z_score
14    300  3.332143
```

c. Interquartile Range (IQR)

IQR adalah metode yang populer untuk mendeteksi outliers. IQR dihitung dengan mengurangi kuartil pertama (Q1) dari kuartil ketiga (Q3). Nilai yang berada di luar Q1 - 1.5 * IQR atau Q3 + 1.5 * IQR dianggap sebagai outliers.

```
data = {'Value': [10, 12, 15, 13, 14, 120, 130, 14, 16, 13, 12, 115, 14, 16]}
df = pd.DataFrame(data)

# Menghitung Q1 (kuartil pertama) dan Q3 (kuartil ketiga)
Q1 = df['Value'].quantile(0.25)
Q3 = df['Value'].quantile(0.75)
IQR = Q3 - Q1

# Mengidentifikasi outliers
outliers = df[(df['Value'] < (Q1 - 1.5 * IQR)) | (df['Value'] > (Q3 + 1.5 * IQR))]
print("Outliers berdasarkan IQR:\n", outliers)
```

```
Outliers berdasarkan IQR:
    Value
5     120
6     130
11    115
```

Kemudian, *outlier* yang telah diidentifikasi dapat ditangani dengan beberapa pendekatan, seperti:

a. Menghapus *outliers* apabila disebabkan oleh kesalahan entri data atau data tidak relevan dengan analisis. Berikut merupakan contoh penghapusan *outlier* berdasarkan Z-Score:

```
df_cleaned = df[df['z_score'] <= 3]
```

b. Mengganti *outliers* dengan nilai yang lebih masuk akal, misalnya nilai batas IQR atau nilai median.
c. Melakukan transformasi data

C. *Data Reduction*

Reduksi data adalah proses mengurangi volume atau dimensi data dengan tetap mempertahankan integritas dan informasi penting. Teknik data reduction dapat diterapkan untuk memperoleh representasi tereduksi dari kumpulan data yang volumenya jauh lebih kecil, namun tetap menjaga integritas data asli. Penambangan pada kumpulan data yang dikurangi harus lebih efisien namun menghasilkan hasil analisis yang sama (atau hampir sama).

1. *Dimensionality reduction*

Reduksi dimensi adalah teknik untuk mengurangi jumlah fitur atau variabel dalam dataset dengan tetap mempertahankan sebagian besar informasi. Pengurangan dimensi kumpulan data mempunyai beberapa manfaat:

- Mengurangi waktu pelatihan, mengurangi kebutuhan memori dan meningkatkan kinerja algoritma *machine learning*
- Menghilangkan atribut yang tidak relevan dan mengurangi jumlah atribut yang bising

- Membuat visualisasi data lebih mudah dipahami dan memungkinkan visualisasi kumpulan data dengan jumlah atribut yang tinggi

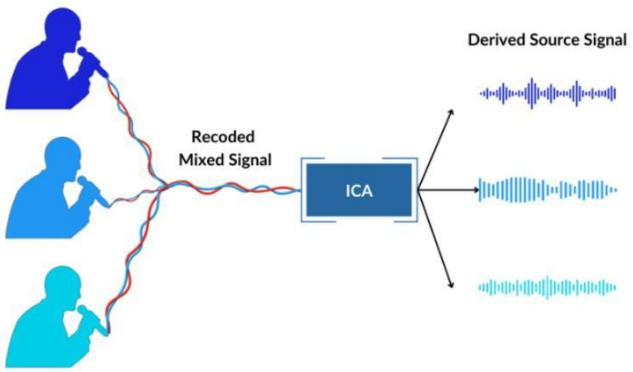
Teknik yang biasa digunakan:

a. ***Principal Component Analysis (PCA)*** yaitu mengubah data ke dalam set komponen utama yang tidak berkorelasi dan mempertahankan variasi maksimum dalam data dengan komponen yang lebih sedikit. Contoh penggunaan pada teknik PCA dapat dilihat pada gambar dibawah ini.

```
from sklearn.decomposition import PCA
import numpy as np

data = [[2, 3], [3, 6], [4, 8], [5, 10], [6, 12]]

# Convert list to numpy array
X = np.array(data)

# Apply PCA to reduce dimensions from 2 to 1
pca = PCA(n_components=1)
X_pca = pca.fit_transform(X)

print("Original Data:\n", X)
print("PCA Transformed Data:\n", X_pca)
```

```
Original Data:
 [[ 2  3]
 [ 3  6]
 [ 4  8]
 [ 5 10]
 [ 6 12]]
PCA Transformed Data:
 [[-5.197773 ]
 [-2.05203067]
 [ 0.18228528]
 [ 2.41660122]
 [ 4.65091717]]
```

b. Independent Component Analysis (ICA)



Independent Component Analysis (ICA) adalah teknik untuk memisahkan data multivariat menjadi komponen-komponen yang independen secara statistik. Dapat dilihat contoh pada gambar diatas menunjukkan tiga orang yang sedang berbicara ke mikrofon dan tiap individunya mewakili sebuah sumber suara yang independen.

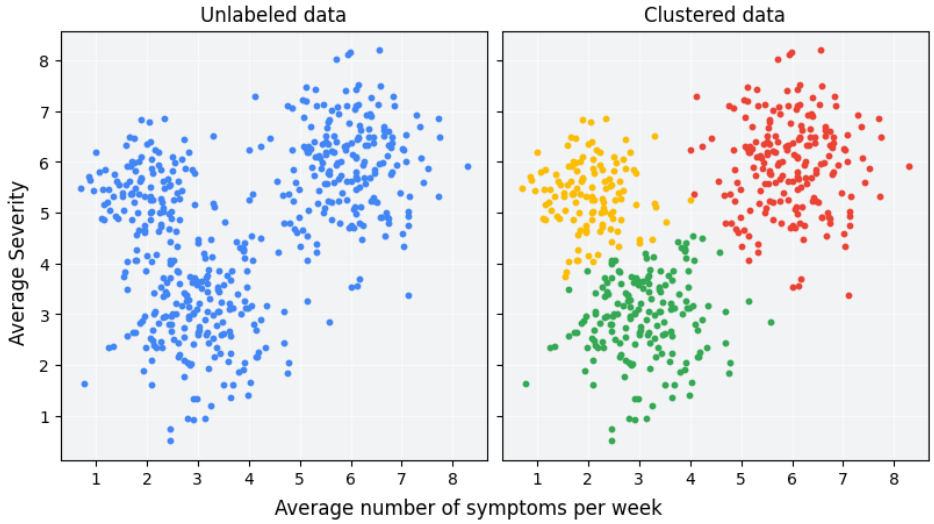
Suara dari ketiga orang tersebut terekam dalam satu rekaman yang sama (Mixed Signal) yang dimana suara dari ketiga sumber saling tumpang tindih dan sulit dibedakan. Sinyal campuran ini kemudian diproses menggunakan ICA yang bekerja dengan mencari komponen-komponen statistik yang paling tidak berkorelasi satu sama lain, sehingga dapat memisahkan suara dari sumber yang berbeda. Setelah melalui proses ICA, sinyal campuran berhasil dipisahkan menjadi tiga sinyal sumber yang independen.

c. ***Multidimensional Scalling (MDS)*** adalah sebuah teknik analisis statistik multivariat yang digunakan untuk menggambarkan struktur hubungan antar objek data berdasarkan kemiripannya secara grafis dalam suatu bidang multidimensi, untuk mendapatkan informasi dari data.

2. *Numerosity Reduction*

Teknik ini mengurangi volume data dengan mengganti data asli dengan representasi yang lebih kecil. Metode-metodenya meliputi:

a. ***Clustering*** adalah proses mengelompokkan data menjadi beberapa kelompok berdasarkan kesamaan. Contoh clustering dapat dilihat pada gambar dibawah ini.



b. ***Sampling*** adalah teknik mengambil sebagian kecil data sebagai representasi dari seluruh data

```
import numpy as np
import pandas as pd

# Generate sample data
np.random.seed(42)
data = np.random.rand(1000, 2)

# Create a Pandas DataFrame
df = pd.DataFrame(data, columns=["Feature 1", "Feature 2"])

# Take a random sample of 10% of the data
sample = df.sample(frac=0.1, random_state=42)

# Print the sample data
print("Sample data:")
print(sample)
```

```
Sample data:
     Feature 1  Feature 2
521   0.318753   0.625891
737   0.327033   0.667593
740   0.817834   0.120209
660   0.247103   0.159545
411   0.333499   0.398169
..         ...        ...
436   0.644000   0.408734
764   0.803026   0.539161
88    0.690938   0.386735
63    0.818015   0.860731
826   0.263113   0.360136

[100 rows x 2 columns]
```

Pada contoh kode diatas menghasilkan 1000 data acak dengan dua fitur yang berbeda, kemudian mengambil sampel data sebesar 10% dari data tersebut secara acak dari keseluruhan data.

D. Handling Missing Value

Missing values merujuk pada kondisi dimana data untuk atribut tertentu tidak tersedia.
Penyebab *Missing Values* diantara lain :

- Kesalahan pengumpulan atau input data
- Nilai atribut tidak diketahui

- Atribut tidak diperlukan untuk objek tertentu
- Adanya penghapusan *outlier* atau data tidak valid
- *Programming error*
- Data hilang saat transfer data secara manual dari *legalcy database.*

1. *Detecting missing values*

Proses *detecting missing values* data dilakukan dengan metode **isnull(), notnull()** dan **info()**

- Mengecek nilai *null*

  isnull() dapat mendeteksi adanya *missing* atau *null* pada dataset serta dapat mengembalikan sebuah boolean *mask*, dimana kondisi **'True'** menunjukkan adanya ***missing value*** dan **'false'** menunjukan ***non-missing value***. **notnull()** memiliki cara kerja yang sama hanya saja kondisi 'False' menunjukkan adanya ***missing value*** dan 'True' menunjukan ***non-missing value.***

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
data = {
    'modul1': ['JESC', 'Fath', 'BETA', 'DataMining'],
    'semester': [7, np.nan, 7, np.nan],
    'Shift': [4, 1, 5, 7 ]
}

df = pd.DataFrame(data)
```

```
missingvalue = df.isnull()
print(missingvalue)
```

```
   modul1  semester  Shift
0   False     False  False
1   False      True  False
2   False     False  False
3   False      True  False
```

```
missingvalue = df.notnull()
print(missingvalue)
```

```
   modul1  semester  Shift
0    True      True   True
1    True     False   True
2    True      True   True
3    True     False   True
```

- Mengecek informasi data

  info() dapat menampilkan jumlah total baris, jumlah *non-null (non-missing values)*, dan tipe data untuk setiap kolom.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
data = {
    'modul1': ['JESC', 'Fath', 'BETA', 'DataMining'],
    'semester': [7, np.nan, 7, np.nan],
    'Shift': [4, 1, 5, 7 ]
}

df = pd.DataFrame(data)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   modul1    4 non-null      object
 1   semester  2 non-null      float64
 2   Shift     4 non-null      int64
dtypes: float64(1), int64(1), object(1)
```

- Visualisasi *missing data* menggunakan *heatmap*

  Memilih untuk menggunakan heatmap bergantung pada kebutuhan dan sifat dataset.

Heatmap paling efektif untuk data numerik kontinu atau diskrit, serta data kategorikal yang telah diringkas. Namun, jika ada pencilan ekstrem atau data yang sangat jarang, heatmap mungkin memerlukan preprocessing atau normalisasi

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
data = {
    'modul1': ['JESC', 'Fath', 'BETA', 'DataMining'],
    'semester': [7, np.nan, 7, np.nan],
    'Shift': [4, 1, 5, 7 ]
}

df = pd.DataFrame(data)
```

```python
# Set ukuran plot
plt.figure(figsize=(8, 4))

# Buat heatmap
sns.heatmap(df.isnull(), cbar=False, cmap='viridis', annot=True)

# Tambahkan judul
plt.title('Visualisasi Missing Data')
plt.xlabel('Kolom')
plt.ylabel('Baris')

# Tampilkan plot
plt.show()
```

2. *Types Of Missing Value*

a. MCAR - Missing completely at random

Terjadi jika semua *variabels* dan *observations* memiliki probabilitas yang sama untuk hilang

b. MAR - Missing at random

Probabilitas hilangnya suatu *value* berkaitan dengan *value* dari variabel itu sendiri atau variabel lain dalam dataset. Sehingga tidak semua *observations* dan *variabels* memiliki peluang yang sama untuk hilang.

c. MNAR - Missing not at random

MNAR adalah tipe ketika MAR atau MCAR tidak berlaku. Dalam situasi ini, probabilitas hilangnya data, berbeda-beda untuk setiap *value* dari variabel yang sama, dan alasan di balik hal ini bisa jadi tidak diketahui

3. *Missing Value Handling*

Terdapat beberapa teknik dalam mengatasi *missing values* yaitu dengan menghapus data yang hilang, imputation nilai rata-rata, median, modus dari variabel terkait, atau dapat juga menggunakan *advanced imputation.*

a. Menghapus data dengan dropna()

Teknik dasar dalam mengatasi *missing value* adala dengan menghapus data yang hilang tersebut. Dapat dilakukan dengan cara menghapus baris atau seluruh kolom yang tidak memiliki nilai (*missing value)*

```
# Menghapus baris dengan missing values
df_dropped_rows = df.dropna()
print(df_dropped_rows)
```

```
  modul1  semester  Shift
0   JESC       7.0      4
2   BETA       7.0      5
```

```
# Menghapus kolom dengan missing values
df_dropped_cols = df.dropna(axis=1)
print(df_dropped_cols)
```

```
         modul1  Shift
0          JESC      4
1          Fath      1
2          BETA      5
3  DataMining        7
```

b. Mengganti nilai pada *missing values*

Fungsi **`fillna()`** dalam pandas digunakan untuk mengganti nilai yang hilang dengan nilai tertentu. Contohnya, kita dapat mengganti nilai yang hilang dengan rata-rata kolom menggunakan **`df.mean()`,** yang berguna untuk data numerik.Menggunakan rata-rata untuk imputasi bisa menimbulkan bias jika tidak mencerminkan distribusi data yang sebenarnya. Sebagai alternatif, dapat menggunakan median (**`df.median()`**), yang lebih stabil terhadap pencilan, atau modus (**`df.mode().iloc[0]`**), yaitu nilai yang paling sering muncul dalam data.

```
# Mengisi dengan nilai rata-rata
df_filled_mean = df.fillna(df.mean())

# Mengisi dengan nilai median
df_filled_median = df.fillna(df.median())

# Mengisi dengan nilai modus
df_filled_mode = df.fillna(df.mode().iloc[0])
```

4. Advance Imputation (KKN imputation)

Teknik berikutnya adalah menggunakan metode imputasi yang lebih kompleks, seperti K-Nearest Neighbors (KNN) atau Regresi.

Metode ini lebih cepat dalam mempertimbangkan nilai-nilai lain di dataset untuk mengestimasi missing values, sehingga hasilnya bisa lebih akurat.

```
from sklearn.impute import KNNImputer

# Inisialisasi KNN Imputer
imputer = KNNImputer(n_neighbors=2)

# Menggunakan KNN untuk mengisi missing values
df_knn_imputed = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)
```

E. Data Transformation

Transformasi data adalah langkah penting dalam tahap sebelum pemrosesan data yang bertujuan untuk mengubah data mentah menjadi format yang sesuai untuk analisis. **Tujuan utama** dari transformasi data adalah untuk meningkatkan kualitas dan efisiensi interpretasi data, pemodelan, dan prediksi. Beberapa langkah utama dalam transformasi data meliputi:

a. Data Encoding

Data *encoding* adalah langkah penting dalam pra-pemrosesan data di *Machine Learning*. Proses ini mengubah data kategorikal atau teks menjadi format numerik, sehingga dapat digunakan sebagai input oleh algoritma machine learning.

Kenapa menggunakan Data Encoding?

- Sebagian besar algoritma machine learning hanya dapat bekerja dengan data numerik, sehingga variabel kategorikal perlu diubah menjadi nilai numerik.
- Encoding membantu model mengenali pola dalam data dan membuat prediksi berdasarkan pola tersebut.
- Encoding juga mencegah bias dalam model dengan memastikan semua fitur memiliki bobot yang setara.
- Pemilihan metode encoding yang tepat dapat mempengaruhi kinerja model secara signifikan.

Metode yang dapat diterapkan untuk melakukan *encoding* pada variabel/atribut kategorikal meliputi: *One-Hot Encoding*, *Dummy Encoding*, *Ordinal Encoding*, *Binary Encoding*, *Count Encoding*, dan *Target Encoding.*

Studi Kasus:

Terdapat dataset tentang pelanggan yang berisi informasi tentang jenis kelamin dan kota tempat tinggal. Variabel-variabel tersebut adalah kategorikal dan perlu di-encode menjadi bentuk numerik sebelum dapat digunakan dalam model machine learning.

- **Nama**: Alice, Bob, Charlie, David, Eva
- **Jenis Kelamin**: Perempuan, Laki-laki, Laki-laki, Laki-laki, Perempuan
- **Kota**: Jakarta, Bandung, Jakarta, Surabaya, Bandung

Contoh Penggunaan:

```
import pandas as pd

# Dataset pelanggan
data = {
    'Nama': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
    'Jenis_Kelamin': ['Perempuan', 'Laki-laki', 'Laki-laki', 'Laki-laki', 'Perempuan'],
    'Kota': ['Jakarta', 'Bandung', 'Jakarta', 'Surabaya', 'Bandung']
}


# Melakukan One-Hot Encoding pada kolom Jenis_Kelamin dan Kota
df_encoded = pd.get_dummies(df, columns=['Jenis_Kelamin', 'Kota'])

df = pd.DataFrame(data)
print("Data Pelanggan Awal:")
print(df)
print("\nData Pelanggan Setelah One-Hot Encoding:")
print(tabulate(df_encoded, headers='keys'))
```

Output:

```
Data Pelanggan Awal:
      Nama Jenis_Kelamin      Kota
0    Alice     Perempuan   Jakarta
1      Bob     Laki-laki   Bandung
2  Charlie     Laki-laki   Jakarta
3    David     Laki-laki  Surabaya
4      Eva     Perempuan   Bandung

Data Pelanggan Setelah One-Hot Encoding:
    Nama     Jenis_Kelamin_Laki-laki    Jenis_Kelamin_Perempuan    Kota_Bandung    Kota_Jakarta    Kota_Surabaya
--  -------  -------------------------  -------------------------  --------------  --------------  ---------------
 0  Alice    False                      True                       False           True            False
 1  Bob      True                       False                      True            False           False
 2  Charlie  True                       False                      False           True            False
 3  David    True                       False                      False           False           True
 4  Eva      False                      True                       True            False           False
```

Penjelasan:

- Pada studi kasus ini, metode **One-Hot Encoding** diterapkan dengan membuat kolom biner untuk setiap kategori unik dalam variabel *Jenis Kelamin* dan *Kota*.
- Pada kolom Jenis_Kelamin, jika pelanggan adalah "Laki-laki", maka kolom Jenis_Kelamin_Laki-laki akan bernilai 1, dan kolom Jenis_Kelamin_Perempuan akan bernilai 0 (dan sebaliknya).
- Pada kolom Kota, setiap kota juga diwakili oleh kolom biner, di mana 1 menunjukkan bahwa pelanggan tinggal di kota tersebut.

b. Agregasi

Agregasi adalah proses menyederhanakan data dengan menggabungkan beberapa nilai atau data menjadi satu nilai yang representatif. Proses ini sering digunakan dalam analisis data skala besar untuk membantu kita memahami pola dan tren dalam data dengan lebih jelas.

Teknik agregasi yang umum meliputi menghitung rata-rata, jumlah, nilai maksimum, nilai minimum, dan banyak lagi.

Contoh Penggunaan:

- **Menghitung Rata-Rata Penjualan**: Jika kita memiliki data penjualan harian untuk beberapa produk, kita bisa menghitung rata-rata penjualan per bulan untuk melihat tren penjualan.
- **Menjumlahkan Pendapatan**: Dalam laporan keuangan, kita sering kali menjumlahkan pendapatan dari berbagai sumber untuk mendapatkan total pendapatan perusahaan.
- **Mengelompokkan Data**: Ketika menganalisis data pelanggan, kita bisa mengelompokkan mereka berdasarkan usia atau lokasi dan kemudian menghitung rata-rata pengeluaran per kelompok.

Studi Kasus:

Untuk studi kasus ini, akan digunakan dataset yang berisi informasi tentang penjualan harian beberapa produk selama bulan September 2024. Dataset ini memiliki tiga kolom utama:

Produk = ['A', 'A', 'B', 'B', 'C', 'C', 'D', 'D']

Tanggal = ['2024-09-01', '2024-09-02', '2024-09-01', '2024-09-02', '2024-09-01', '2024-09-02', '2024-09-01', '2024-09-02']

Jumlah Penjualan = [10, 15, 5, 10, 8, 12, 20, 30]

Dalam studi kasus ini, tujuan kita adalah untuk menghitung total penjualan untuk setiap produk selama periode yang ditentukan (dua hari dalam dataset).

Contoh Penggunaan:

```
import pandas as pd

# Membuat dataset
data = {
    'produk': ['A', 'A', 'B', 'B', 'C', 'C', 'D', 'D'],
    'tanggal': ['2024-09-01', '2024-09-02', '2024-09-01', '2024-09-02', '2024-09-01', '2024-09-02', '2024-09-01', '2024-09-02'],
    'jumlah_penjualan': [10, 15, 5, 10, 8, 12, 20, 30]
}

df = pd.DataFrame(data)

# Mengubah kolom tanggal menjadi format datetime
df['tanggal'] = pd.to_datetime(df['tanggal'])

# Agregasi data: Hitung total penjualan per produk
total_penjualan = df.groupby('produk')['jumlah_penjualan'].sum().reset_index()

# Tampilkan hasil
print("Dataset Awal:")
print(df)
print("\nTotal Penjualan Per Produk:")
print(total_penjualan)
```

Output:

```
Dataset Awal:
  produk    tanggal  jumlah_penjualan
0      A 2024-09-01                10
1      A 2024-09-02                15
2      B 2024-09-01                 5
3      B 2024-09-02                10
4      C 2024-09-01                 8
5      C 2024-09-02                12
6      D 2024-09-01                20
7      D 2024-09-02                30

Total Penjualan Per Produk:
  produk  jumlah_penjualan
0      A                25
1      B                15
2      C                20
3      D                50
```

c. *Normalization (min-max)*

Normalisasi adalah proses yang bertujuan untuk mengubah data ke dalam rentang atau skala tertentu agar lebih mudah dianalisis dan dibandingkan. Normalisasi penting dalam banyak aplikasi analisis data, terutama ketika bekerja dengan algoritma yang sensitif terhadap skala data, seperti algoritma pembelajaran mesin. Teknik normalisasi yang paling umum adalah ***Min-Max Scaler***.

- **Min-Max Scaler**

  Min-Max Scaler mengubah nilai data menjadi rentang baru, biasanya dari 0 hingga 1. Ini berguna ketika kita ingin memastikan semua fitur memiliki skala yang sama. Rumus:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Dimana:

- x adalah nilai asli data.
- (x) adalah nilai minimum dalam dataset.
- (x) adalah nilai maksimum dalam dataset.
- x adalah nilai data yang telah dinormalisasi.

Studi Kasus:

Misalkan kita memiliki dataset yang berisi nilai tinggi badan dalam cm dari beberapa individu:

$Tinggi\ Badan$: [150, 160, 170, 180, 190]

Contoh Penggunaan:

```
import pandas as pd

# Dataset tinggi badan (dalam cm)
data = {'Tinggi_Badan': [150, 160, 170, 180, 190]}
df = pd.DataFrame(data)

# Normalisasi Min-Max
df['Tinggi_Badan_Normalized'] = (df['Tinggi_Badan'] - df['Tinggi_Badan'].min()) / (df['Tinggi_Badan'].max() - df['Tinggi_Badan'].min())

print("Data Tinggi Badan Awal:")
print(df[['Tinggi_Badan']])
print("\nData Tinggi Badan Setelah Normalisasi Min-Max:")
print(df[['Tinggi_Badan_Normalized']])
```

Output:

```
Data Tinggi Badan Awal:
   Tinggi_Badan
0           150
1           160
2           170
3           180
4           190

Data Tinggi Badan Setelah Normalisasi Min-Max:
   Tinggi_Badan_Normalized
0                     0.00
1                     0.25
2                     0.50
3                     0.75
4                     1.00
```

- **Tinggi_Badan_Normalized**: Nilai tinggi badan yang dinormalisasi menggunakan Min-Max Normalization, yang akan berada dalam rentang 0 hingga 1.

d. Standardization (Z-score)

Standardization yang biasanya menggunakan metode Z-Score adalah teknik yang menskalakan data berdasarkan distribusi statistiknya, yakni rata-rata (mean) dan deviasi standar. Nilai hasil Z-Score menunjukkan seberapa jauh sebuah data dari rata-rata dalam satuan deviasi standar. Teknik ini sering digunakan ketika kita ingin membandingkan data dalam distribusi normal atau untuk analisis statistik.

Rumus :

$$z = \frac{x - \mu}{\sigma}$$

- x adalah nilai asli data.
- μ adalah rata-rata dari data.
- σ adalah deviasi standar dari data.
- z adalah nilai yang telah di standardisasi dalam bentuk Z-Score

Contoh Penggunaan:

((Menggunakan studi kasus yang sama dengan *Min-Max Scaler*))

```
# Menghitung rata-rata dan deviasi standar
mean = df['Tinggi_Badan'].mean()
std_dev = df['Tinggi_Badan'].std()

# Normalisasi Z-Score
df['Tinggi_Badan_ZScore'] = (df['Tinggi_Badan'] - mean) / std_dev

print("\nData Tinggi Badan Setelah Normalisasi Z-Score:")
print(df[['Tinggi_Badan_ZScore']])
```

Output:

```
Data Tinggi Badan Setelah Normalisasi Z-Score:
   Tinggi_Badan_ZScore
0            -1.264911
1            -0.632456
2             0.000000
3             0.632456
4             1.264911
```

- **Tinggi_Badan_ZScore**: Nilai tinggi badan yang dinormalisasi menggunakan Z-Score Normalization, yang menunjukkan seberapa banyak setiap nilai deviasi dari rata-rata.

e. Discretization

Discretization adalah proses mengubah data kontinu menjadi data diskrit dengan cara membagi data ke dalam interval tertentu. Proses ini sering digunakan dalam analisis data dan pemodelan untuk mengurangi kompleksitas data, serta membuatnya lebih mudah diinterpretasikan. *Discretization* membantu dalam mengelompokkan nilai-nilai yang mendekati satu sama lain ke dalam kategori yang sama.

Studi Kasus:

Terdapat dataset yang berisi informasi tentang pendapatan individu dalam dolar. Akan dilakukan pengelompokan data pendapatan ke dalam beberapa kategori: "Rendah", "Sedang", dan "Tinggi".

$$Pendapatan = [20000, 35000, 50000, 75000, 100000, 150000,$$

Contoh Penggunaan:

```python
import pandas as pd

# Dataset pendapatan
data = {'Pendapatan': [20000, 35000, 50000, 75000, 100000, 150000, 200000]}
df = pd.DataFrame(data)

print("Data Pendapatan Awal:")
print(df)

# Mendiskritkan data pendapatan
bins = [0, 40000, 100000, float('inf')]  # Batasan interval
labels = ['Rendah', 'Sedang', 'Tinggi']  # Nama kategori

df['Kategori'] = pd.cut(df['Pendapatan'], bins=bins, labels=labels, right=False)

print("\nData Pendapatan Setelah Discretization:")
print(df)
```

Output:

```
Data Pendapatan Awal:
   Pendapatan
0       20000
1       35000
2       50000
3       75000
4      100000
5      150000
6      200000
```

```
Data Pendapatan Setelah Discretization:
   Pendapatan Kategori
0       20000   Rendah
1       35000   Rendah
2       50000   Sedang
3       75000   Sedang
4      100000   Tinggi
5      150000   Tinggi
6      200000   Tinggi
```

SUMBER

https://www.codecademy.com/article/eda-data-visualization

https://medium.com/aiskunks/categorical-data-encoding-techniques-

d629669740f https://www.datacamp.com/tutorial/techniques-to-handle-missing-

data-values

https://towardsdatascience.com/missing-value-imputation-explained-a-visual-guide-with-code-exampl es-for-beginners-93e0726284eb

https://machinelearningtutorials.org/pandas-isnull-function-explained-with-examples/