



MACHINE LEARNING

Transformasi Data

Magister Teknik Informatika

Dr. Chairani, S.Kom., M.Eng

IIB DARMAJAYA, 2023/2024

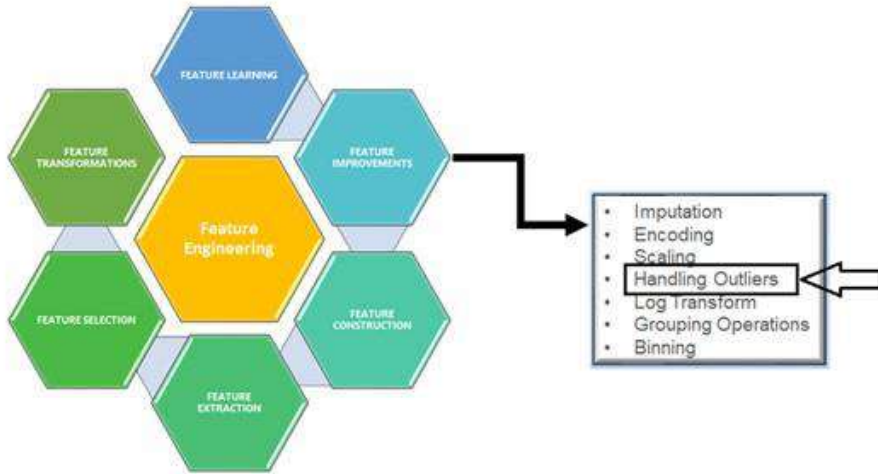
Subject

- Kursus ini adalah bagian dari Data Preparation
- Data Preparation yang dibahas adalah transformasi data yaitu:
 - Representasi Fitur, dan
 - Rekayasa Fitur
- Ada beberapa teknik transformasi data yang digunakan sesuai kebutuhan dan ketersediaan/jenis data baik numerik maupun kategorik

Data Transformation

- Representasi Fitur atau Pembelajaran Fitur:
 - Teknik-Teknik yang memungkinkan sistem bekerja otomatis menemukan representasi yang diperlukan (untuk deteksi fitur atau klasifikasi dari dataset),
 - menggantikan rekayasa fitur manual, dan
 - memungkinkan mesin mempelajari fitur dan menggunakannya untuk melakukan tugas tertentu.
- Rekayasa Fitur:
 - Proses mengubah data mentah menjadi fitur yang:
 - Mewakili masalah mendasar ke model prediktif,
 - menghasilkan akurasi model yang lebih baik pada data yang tidak terlihat.

Rekayasa Fitur

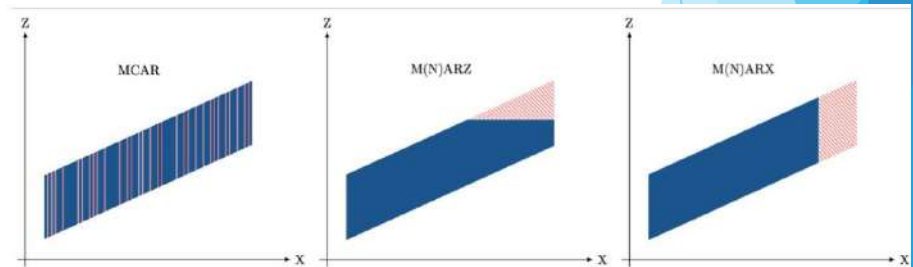
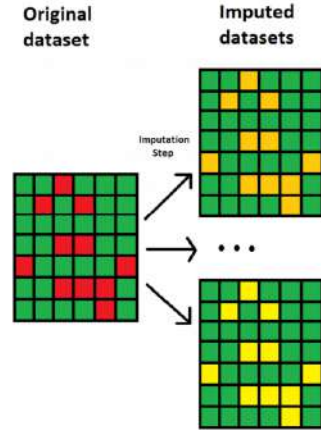


Outlier



Imputasi

- Pengertian: Mengganti nilai/data yang hilang (missing value; NaN; blank) dengan nilai pengganti.
- Jenis missing value:
 1. Missing Completely At Random (MCAR).
 2. Missing At Random (MAR).
 3. Missing Not At Random (MNAR).



sum. gbr:
<https://www.ud.ac.uk/population-health-sciences/sites/population-health-sciences>
https://rianneschouten.github.io/missing_data_science/assets/blogpost/blogpost.html

Missing At Random (MAR)

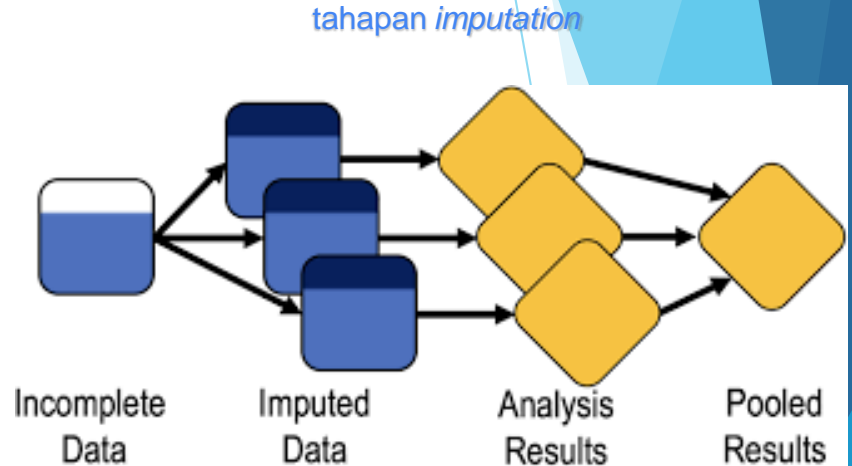
- Definisi: Probabilitas sebuah instance yang hilang mungkin bergantung pada nilai yang diketahui tetapi tidak pada nilai yang hilang itu sendiri.
- Contoh Sensor: Dalam kasus sensor suhu, fakta bahwa suatu nilai hilang tidak bergantung pada suhu, tetapi mungkin bergantung pada beberapa faktor lain, misalnya pada daya baterai termometer.
- Contoh survei: Apakah seseorang menjawab pertanyaan atau tidak - mis. tentang usia- dalam survei tidak tergantung pada jawaban itu sendiri, tetapi mungkin tergantung pada jawaban untuk pertanyaan lain, yaitu jenis kelamin perempuan.

Missing Not At Random (MNAR)

- Definisi: probabilitas sebuah instance hilang dapat bergantung pada nilai variabel itu sendiri.
- Contoh sensor: Dalam kasus sensor suhu, sensor tidak berfungsi dengan baik saat suhu lebih dingin dari 5°C .
- Contoh survei: Apakah seseorang menjawab pertanyaan atau tidak - mis. jumlah hari sakit - dalam survei memang tergantung pada jawabannya sendiri - seperti yang bisa terjadi pada beberapa orang yang kelebihan berat badan.

Tahapan dan Teknik Imputasi

- Jika tipe data Variabel Numerik
 - Imputasi mean atau median.
 - Imputasi nilai suka-suka (arbitrary).
 - Imputasi nilai/data ujung (end of tail).
- Jika tipe data Variabel Kategorik
 - Imputasi kategori yang sering muncul.
 - Tambah kategori yang hilang.



Imputasi Mean atau Median

- Pro:
 - Mudah dan cepat.
 - Bekerja efektif untuk dataset numerik berukuran kecil.
 - Cocok untuk variabel numerik.
 - Cocok untuk data missing completely at random (MCAR)
 - Dapat digunakan dalam produksi (mis. dalam model deployment)
- Kontra:
 - Tidak memperhitungkan korelasi antar fitur, berfungsi pada tingkat kolom.
 - Kurang akurat.
 - Tidak memperhitungkan probabilitas/ketidakpastian.
 - Tidak cocok utk >5%missing data.
 - Mendistorsi variansi dan distribusi variabel asal/orijinal serta covariant variabel sisa data.



	col1	col2	col3	col4	col5		col1	col2	col3	col4	col5	
0	2	5.0	3.0	6	NaN	mean()	0	2.0	5.0	3.0	6.0	7.0
1	9	NaN	9.0	0	7.0		1	9.0	11.0	9.0	0.0	7.0
2	19	17.0	NaN	9	NaN		2	19.0	17.0	6.0	9.0	7.0

Age		Age
29	$\begin{aligned} \text{mean} \\ &= (29 + 43 + 25 + 34 + 50)/5 \\ &= 36,2 \end{aligned}$	29
43		43
NA		36.2
25		25
34		34
NA		36.2
50		50

sumber gambar

<https://towardsdatascience.com/5-different-ways-to-compensate-for-missing-values-data-imputation-with-examples-6022d9ca0779>
<https://heartbeat.fritz.ai/hands-on-with-feature-engineering-techniques-imputing-missing-values-6c22b49c4060>

Hands On

```
import pandas as pd
import numpy as np
```

Import pandas dan numpy

```
kolom = {'col1' : [2, 9, 19],
         'col2' : [5, np.nan, 17],
         'col3' : [3, 9, np.nan],
         'col4' : [6, 0, 9],
         'col5' : [np.nan, 7, np.nan]}
```

Masukkan data

```
data = pd.DataFrame(kolom)
```

pd.DataFrame() --> fungsi mengubah menjadi dataframe

data

	col1	col2	col3	col4	col5
0	2	5.0	3.0	6	NaN
1	9	NaN	9.0	0	7.0
2	19	17.0	NaN	9	NaN

Output

Hands On (Lanjutan)

```
data.fillna(data.mean())
```

Mengganti missing value dengan mean()

	col1	col2	col3	col4	col5
0	2	5.0	3.0	6	7.0
1	9	11.0	9.0	0	7.0
2	19	17.0	6.0	9	7.0

Output

```
umur = {'umur' : [29, 43, np.nan, 25, 34, np.nan, 50]}
```

Masukan Data

```
data = pd.DataFrame(umur)
```

Ubah ke dataframe

```
data
```

	umur
0	29.0
1	43.0
2	NaN
3	25.0
4	34.0
5	NaN
6	50.0

Output

Mengganti missing value dengan mean pada kolom umur

```
data.fillna(data.mean())
```

data.fillna() --> fungsi mengganti missing value

	umur
0	29.0
1	43.0
2	36.2
3	25.0
4	34.0
5	36.2
6	50.0

Output

Imputasi Nilai Suka-suka

- Pro:
 - Asumsi data tidak *missing at random*.
 - Mudah dan cepat diterapkan untuk melengkapi dataset.
- Kontra:
 - Mendistorsi variansi dan distribusi variabel asal/orijinal serta covariant variabel sisa data.
 - Membentuk outlier (jika nilai arbitrer berada di akhir distribusi).
 - Semakin besar NA maka semakin besar distorsi.
 - Hindari memilih nilai arbitrer yg mendekati nilai mean atau median.



Age	Age
29	29
43	43
NA	99
25	25
34	34
NA	99
50	50

Hands On

```
import pandas as pd  
import numpy as np
```

Import pandas dan numpy

```
umur = {'umur' : [29, 43, np.nan, 25, 34, np.nan, 50]}
```

Masukan Data

```
data = pd.DataFrame(umur)  
data
```

Ubah ke dataframe

	umur
0	29.0
1	43.0
2	NaN
3	25.0
4	34.0
5	NaN
6	50.0

Output

```
data.fillna(99)
```

Mengatasi nilai missing value dengan imputasi suka-suka

	umur
0	29.0
1	43.0
2	99.0
3	25.0
4	34.0
5	99.0
6	50.0

Output

Imputasi *End of Tail*

- Pro:
 - Mirip dengan imputasi *suka-suka*.
 - Cocok untuk variabel numerik.
- Ketentuan khusus:
 - Dalam memilih nilai arbitrer:
 - Jika variabel berdistribusi normal, maka nilai arbitrer = $\text{mean} + 3 * \text{std deviasi}$.
 - Jika variabelnya *skew*, maka gunakan aturan IQR proximity (IQR = 75th Quantile - 25th Quantile; Upper limit = 75th Quantile + IQR \times 3; Lower limit = 25th Quantile - IQR \times 3).
 - Hanya digunakan pada data latih (training set).

AGE
29
43
NA
25
34
NA
50



AGE
29
43
66.89
25
34
66.89
50

untuk data distribusi normal

Hands On

```
import pandas as pd  
import numpy as np
```

import pandas
dan numpy

```
umur = {'umur' : [29, 43, np.nan, 25, 34, np.nan, 50]}  
data = pd.DataFrame(umur)  
data
```

masukkan
data

	umur
0	29.0
1	43.0
2	NaN
3	25.0
4	34.0
5	NaN
6	50.0

Output

```
from feature_engine.imputation import EndTailImputer
```

import
EndTail
mputer

```
imputer = EndTailImputer(imputation_method='gaussian', tail='right')
```

membuat
imputer

```
imputer.fit(data)
```

cocokkan imputer ke set

```
test_t = imputer.transform(data)
```

mengubah data

```
test_t
```

	umur
0	29.000000
1	43.000000
2	66.896905
3	25.000000
4	34.000000
5	66.896905
6	50.000000

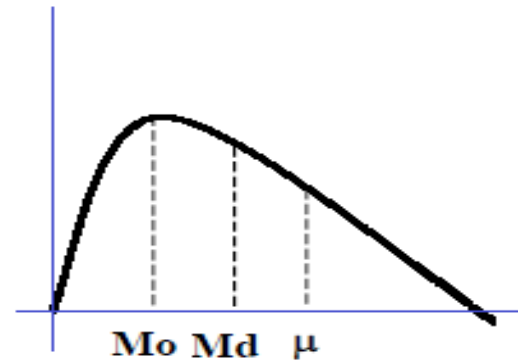
Output

Skewness (kecenderungan/kemiringan/kemencengan)

- Miring ke kanan (positif skew)

Data yang miring ke kanan memiliki ekor panjang yang memanjang ke kanan. Cara alternatif untuk membicarakan kumpulan data yang miring ke kanan adalah dengan mengatakan bahwa itu secara positif.

Jika rata-rata (mean) $>$ median $>$ modus, maka pada kurva distribusi frekuensi, nilai mean akan terletak di sebelah kanan, sedangkan median terletak di tengahnya dan modus di sebelah kiri

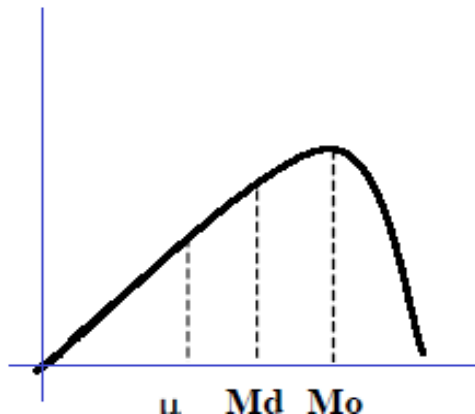


Skewness (kecenderungan/kemiringan/kemencengan)

- Miring ke kiri (negatif skew)

Data yang miring ke kiri memiliki ekor panjang yang memanjang ke kiri. Cara alternatif untuk membicarakan kumpulan data yang miring ke kiri adalah dengan mengatakan bahwa itu miring secara negatif.

Jika rata-rata (mean) < median < modus, maka pada kurva distribusi frekuensi, nilai mean akan terletak di sebelah kiri, sedangkan median terletak di tengahnya dan modus di sebelah kanan

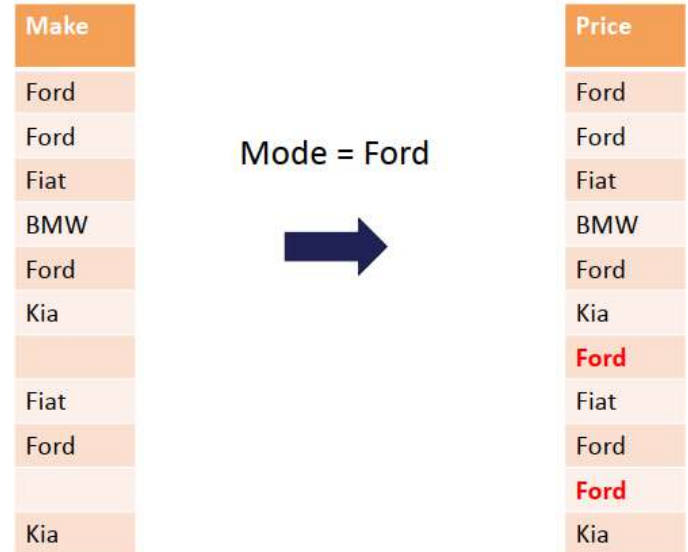


Imputasi *Frequent Category*/Modus

- Pro:
 - Cocok untuk data dengan missing at random.
 - Mudah dan cepat diterapkan.
 - Cocok utk data yang memiliki *skew*
 - Dapat digunakan dalam produksi (mis. dalam model deployment).
- Kontra:
 - Mendistorsi relasi label dengan frekuensi tertinggi vs variabel lain.
 - Menghasilkan over-representation jika banyak data yang missing.

Make	Price
Ford	Ford
Ford	Ford
Fiat	Fiat
BMW	BMW
Ford	Ford
Kia	Kia
	Ford
Fiat	Fiat
Ford	Ford
	Ford
Kia	Kia

Mode = Ford



Hands On

```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
```

Import pandas, SimpleImputer dari
sklearn.impute, numpy

```
make = {'make' : ['Ford', 'Ford', 'Fiat', 'BMW', 'Ford', 'Kia', np.nan, 'Fiat', 'Ford', np.nan, 'Kia']}
```

Masukan
Data

```
data = pd.DataFrame(make)
```

Ubah menjadi data frame

```
data
```

	make
0	Ford
1	Ford
2	Fiat
3	BMW
4	Ford
5	Kia
6	NaN
7	Fiat
8	Ford
9	NaN
10	Kia

Output

Hands On (Lanjutan)

```
imp = SimpleImputer(strategy='most_frequent')
```

```
imp.fit_transform(data)
```

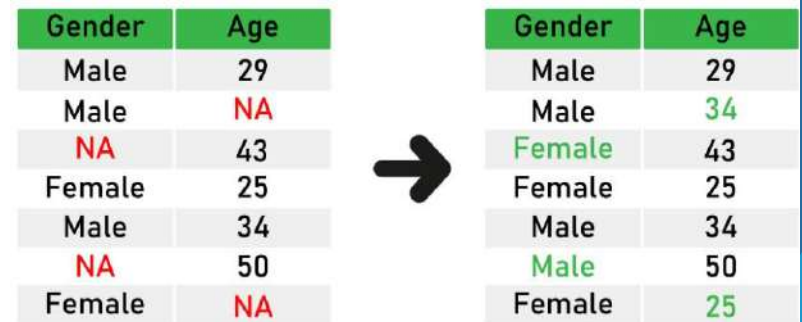
```
array([[ 'Ford'],  
       [ 'Ford'],  
       [ 'Fiat'],  
       [ 'BMW'],  
       [ 'Ford'],  
       [ 'Kia'],  
       [ 'Ford'],  
       [ 'Fiat'],  
       [ 'Ford'],  
       [ 'Ford'],  
       [ 'Kia']], dtype=object)
```

Mengatasi missing value dengan
frequent category / modus

Output

Imputasi *Random Sample*

- Pro:
 - Cocok untuk data *missing at random*.
 - Ganti missing value dengan nilai lain dalam distribusi yang sama dari variabel asli.
 - Mudah dan cepat.
 - Mempertahankan varians dari variabel.
- Kontra:
 - Randomness.
 - Membutuhkan memori besar untuk deployment karena perlu untuk menyimpan data latih yg asli untuk ekstraksi nilai.



Gender	Age
Male	29
Male	NA
NA	43
Female	25
Male	34
NA	50
Female	NA

Gender	Age
Male	29
Male	34
Female	43
Female	25
Male	34
Male	50
Female	25

Hands On

```
from feature_engine.imputation import RandomSampleImputer
import pandas as pd
import numpy as np
```

import randomsample
imputer, pandas, numpy

```
data = {'Gender' : ['Male', 'Male', np.nan, 'Female', 'Male', np.nan, 'Female'],
        'Age' : [29, np.nan, 43, 25, 34, 50, np.nan]}
```

```
df = pd.DataFrame(data)
```

masukkan data dan ubah
menjadi dataframe

```
df
```

	Gender	Age
0	Male	29.0
1	Male	NaN
2	NaN	43.0
3	Female	25.0
4	Male	34.0
5	NaN	50.0
6	Female	NaN

Output

Hands On (Lanjutan)

```
imputer = RandomSampleImputer(random_state = 29)
```



Buat imputernya

```
imputer.fit(df)
```



Cocokkan imputer ke set

```
test_t = imputer.transform(df)
```



Mengubah data

```
test_t
```

	Gender	Age
0	Male	29.0
1	Male	34.0
2	Male	43.0
3	Female	25.0
4	Male	34.0
5	Female	50.0
6	Female	50.0



Output

Imputasi Nilai Nol/Konstanta

- Pro:
 - Cocok untuk variabel kategorik.
- Kontra:
 - Tidak memperhitungkan korelasi antar fitur.
 - Memunculkan bias dalam data.

	col1	col2	col3	col4	col5		col1	col2	col3	col4	col5		
	0	2	5.0	3.0	6	NaN	df.fillna(0)	0	2	5.0	3.0	6	0.0
	1	9	NaN	9.0	0	7.0		1	9	0.0	9.0	0	7.0
	2	19	17.0	NaN	9	NaN		2	19	17.0	0.0	9	0.0

Hands On

```
#Contoh 1 - Imputasi Nilai Nol/Konstanta
#Masukkan data

umur = {'umur' : [29, 43, np.nan, 25, 34, np.nan, 50]}

#Ubah ke dataframe

data = pd.DataFrame(umur)

#Tampilkan nilai yang ada pada variabel 'umur' dalam bentuk dataframe 'data'
```

masukkan data dan ubah menjadi dataframe

```
data
```

	umur
0	29.0
1	43.0
2	NaN
3	25.0
4	34.0
5	NaN
6	50.0

Contoh nilai hilang

```
#Contoh 1 - Imputasi Nilai Nol/Konstanta
#Mengatasi nilai missing value dengan imputasi suka-suka
#Nilai suka-suka yang diberikan pada contoh adalah 99
```

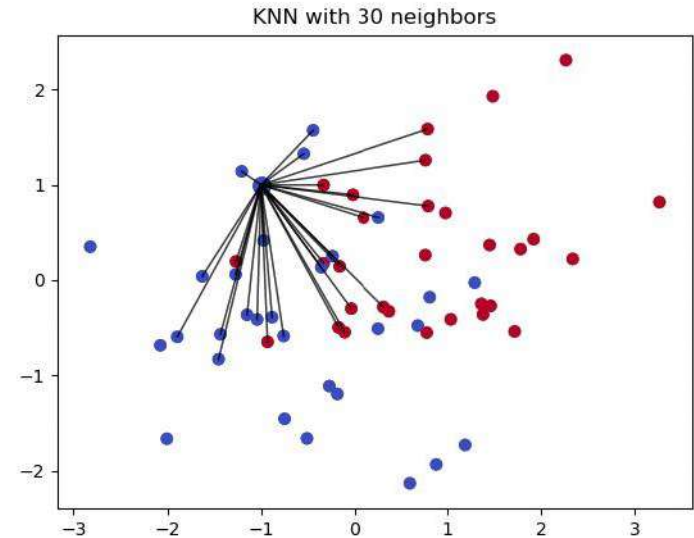
```
data.fillna(0) → Imputasi nilai nol
```

	umur
0	29.0
1	43.0
2	0.0
3	25.0
4	34.0
5	0.0
6	50.0

Output

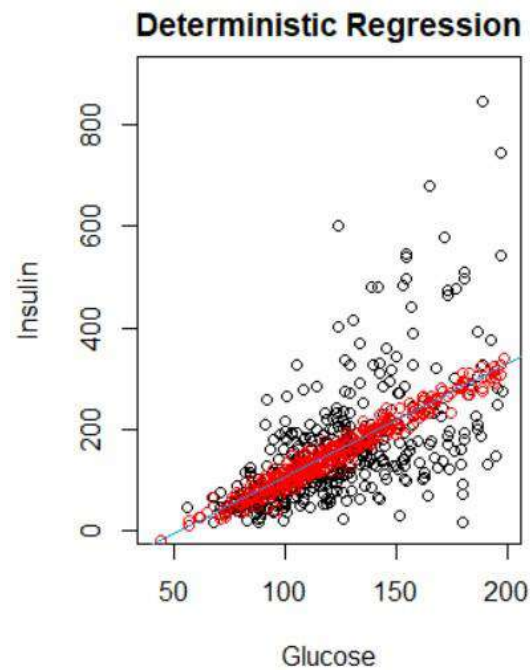
Imputasi dengan K-NN

- Pro:
 - Lebih akurat vs *mean/median/most frequent*.
- Kontra:
 - Biaya komputasi mahal (karena KNN bekerja dengan menyimpan seluruh dataset pelatihan dalam memori).
 - Sensitif terhadap outlier dalam data (tidak seperti SVM).



Imputasi Regresi: Deterministik

- Deterministik
 - Mengganti missing value dengan prediksi yang tepat dari model regresi.
 - Tidak mempertimbangkan variasi acak di sekitar kemiringan (slope regresi).
 - Nilai imputasi seringkali terlalu tepat (precise) dan overestimasi dari korelasi X-Y.



Hands On

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as mno
from sklearn import linear_model
%matplotlib inline
```

→ Import Library

```
df = pd.read_csv("diabetes.csv")
df.head(3)
```

→ Menentukan nilai yang hilang

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1

→ Output

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Pregnancies           768 non-null    int64
1   Glucose                768 non-null    int64
2   BloodPressure         768 non-null    int64
3   SkinThickness         768 non-null    int64
4   Insulin               768 non-null    int64
5   BMI                   768 non-null    float64
6   DiabetesPedigreeFunction 768 non-null    float64
7   Age                   768 non-null    int64
8   Outcome               768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

→ Output

`df.describe()` → menghitung nilai statistik

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

→ Output

```
df.loc[df["Glucose"] == 0.0, "Glucose"] = np.NaN
df.loc[df["BloodPressure"] == 0.0, "BloodPressure"] = np.NaN
df.loc[df["SkinThickness"] == 0.0, "SkinThickness"] = np.NaN
df.loc[df["Insulin"] == 0.0, "Insulin"] = np.NaN
df.loc[df["BMI"] == 0.0, "BMI"] = np.NaN
```

→ `df.loc[]` --> memilih table berdasarkan lokasi

```
df.isnull().sum()[1:6]
```

```
Glucose           5
BloodPressure     35
SkinThickness    227
Insulin          374
BMI              11
dtype: int64
```

→ Output

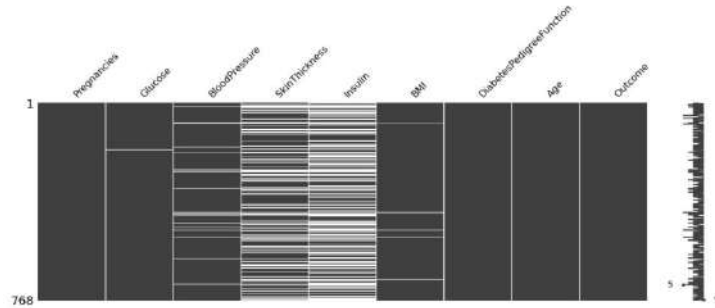
Hands On (Lanjutan)

```
mno.matrix(df, figsize = (20, 6))
```



Membuat Matriks,
figsize --> ukuran figure

<AxesSubplot>



Output

```
missing_columns = ["Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]
```

```
def random_imputation(df, feature):
```

```
    number_missing = df[feature].isnull().sum()
    observed_values = df.loc[df[feature].notnull(), feature]
    df.loc[df[feature].isnull(), feature + '_imp'] = np.random.choice(observed_values, number_missing, replace = True)
    return df
```

```
for feature in missing_columns:
    df[feature + '_imp'] = df[feature]
    df = random_imputation(df, feature)
```



Menggunakan Regresi untuk
memperhitungkan data yang
hilang

Hands On (Lanjutan)

```
deter_data = pd.DataFrame(columns = ["Det" + name for name in missing_columns])  
  
for feature in missing_columns:  
  
    deter_data["Det" + feature] = df[feature + "_imp"]  
    parameters = list(set(df.columns) - set(missing_columns) - {feature + '_imp'})
```

Deterministic
Regression Imputation

```
model = linear_model.LinearRegression()  
model.fit(X = df[parameters], y = df[feature + '_imp'])
```

Buat model Regresi Linier untuk
memperkirakan data yang hilang

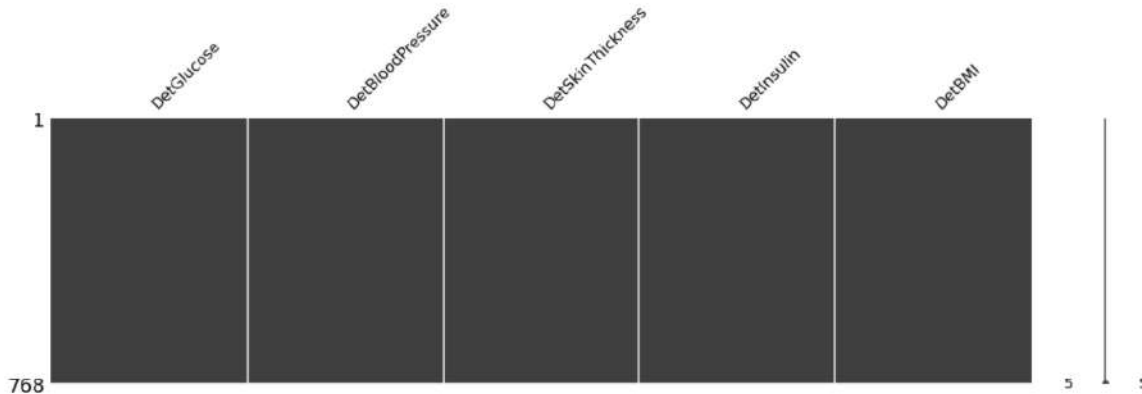
```
deter_data.loc[df[feature].isnull(), "Det" + feature] = model.predict(df[parameters])[df[feature].isnull()]
```

amati bahwa kita
menyimpan indeks data
yang hilang dari kerangka
data asli

```
mno.matrix(deter_data, figsize = (20,5))
```

membuat matriks

<AxesSubplot:>



Hands On (Lanjutan)

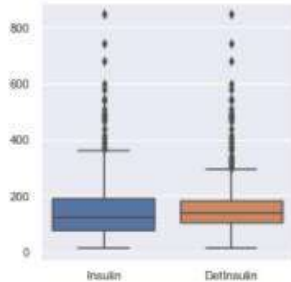
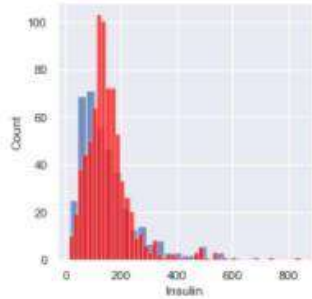
```
sns.set()
fig, axes = plt.subplots(nrows = 2, ncols = 2)
fig.set_size_inches(8, 8)

for index, variable in enumerate(["Insulin", "SkinThickness"]):
    sns.histplot(df[variable].dropna(), kde = False, ax = axes[index, 0])
    sns.histplot(deter_data["Det" + variable], kde = False, ax = axes[index, 0], color = 'red')

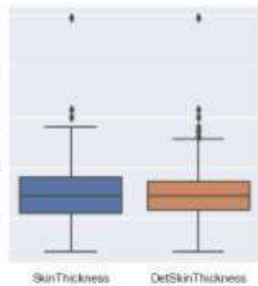
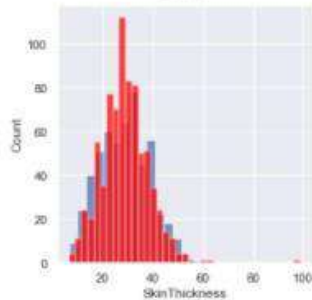
    sns.boxplot(data = pd.concat([df[variable], deter_data["Det" + variable]], axis = 1),
                ax = axes[index, 1])

plt.tight_layout()
```

membuat
chart



Output



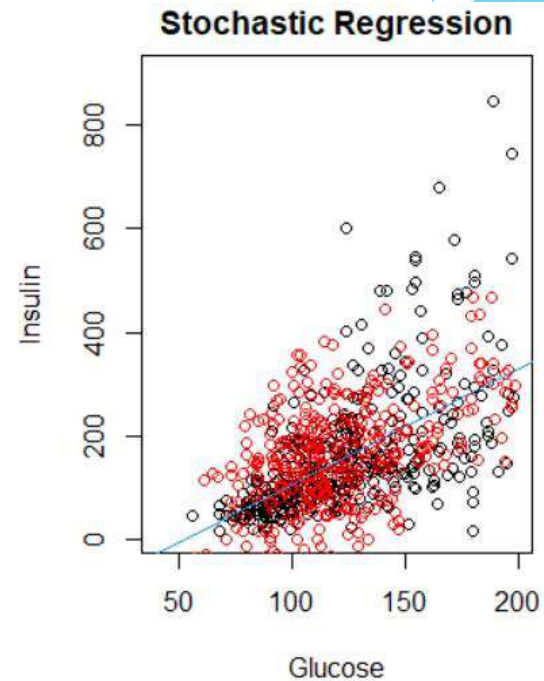
```
pd.concat([df[["Insulin", "SkinThickness"]], deter_data[["DetInsulin", "DetSkinThickness"]]], axis = 1).describe().T
```

	count	mean	std	min	25%	50%	75%	max
Insulin	394.0	155.548223	118.775855	14.0	76.250000	125.000000	190.000000	846.0
SkinThickness	541.0	29.153420	10.476982	7.0	22.000000	29.000000	36.000000	99.0
DetInsulin	768.0	154.559890	89.394698	14.0	105.029991	140.217157	182.000000	846.0
DetSkinThickness	768.0	29.097247	9.189058	7.0	23.000000	29.000000	34.461758	99.0

Output

Imputasi Regresi: Stokastik

- Stokastik
 - Mengatasi masalah dalam imputasi regresi deterministik.
 - Menambahkan fitur “random error” sehingga reproduksi korelasi X-Y lebih tepat.



Hands On

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as mno
from sklearn import linear_model
%matplotlib inline
```

import library

```
df = pd.read_csv("diabetes.csv")
df.head(3)
```

Menentukan nilai yang hilang

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1

Output

```
df.info()
```

melihat info dari data frame

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Pregnancies           768 non-null   int64
1   Glucose               768 non-null   int64
2   BloodPressure         768 non-null   int64
3   SkinThickness         768 non-null   int64
4   Insulin               768 non-null   int64
5   BMI                   768 non-null   float64
6   DiabetesPedigreeFunction 768 non-null   float64
7   Age                   768 non-null   int64
8   Outcome               768 non-null   int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

Output

df.describe() → menghitung data statistik

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Output

```
df.loc[df["Glucose"] == 0.0, "Glucose"] = np.NaN
df.loc[df["BloodPressure"] == 0.0, "BloodPressure"] = np.NaN
df.loc[df["SkinThickness"] == 0.0, "SkinThickness"] = np.NaN
df.loc[df["Insulin"] == 0.0, "Insulin"] = np.NaN
df.loc[df["BMI"] == 0.0, "BMI"] = np.NaN
```

df.loc[] --> memilih tabel berdasarkan lokasi

```
df.isnull().sum()[1:6]
```

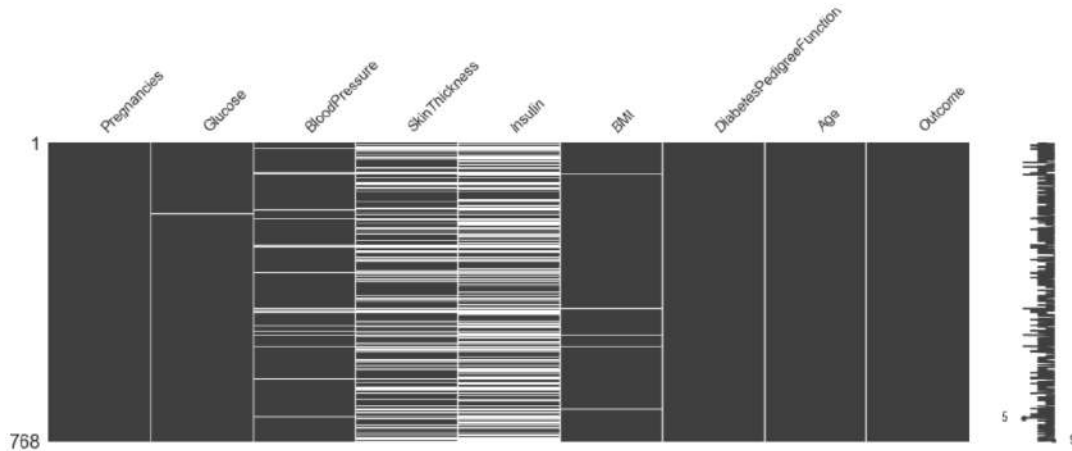
```
Glucose      5
BloodPressure 35
SkinThickness 227
Insulin      374
BMI          11
dtype: int64
```

Output

Hands On (Lanjutan)

```
mno.matrix(df, figsize = (20, 6))
```

```
<AxesSubplot:>
```



→ membuat matriks

→ Output

```
missing_columns = ["Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]
```

```
def random_imputation(df, feature):  
    number_missing = df[feature].isnull().sum()  
    observed_values = df.loc[df[feature].notnull(), feature]  
    df.loc[df[feature].isnull(), feature + '_imp'] = np.random.choice(observed_values, number_missing, replace = True)  
    return df
```

```
for feature in missing_columns:  
    df[feature + '_imp'] = df[feature]  
    df = random_imputation(df, feature)
```

→ Menggunakan Regresi untuk memperhitungkan data yang hilang

Hands On (Lanjutan)

```
random_data = pd.DataFrame(columns = ["Ran" + name for name in missing_columns])  
  
for feature in missing_columns:  
  
    random_data["Ran" + feature] = df[feature + '_imp']  
    parameters = list(set(df.columns) - set(missing_columns) - {feature + '_imp'})  
  
    model = linear_model.LinearRegression()  
    model.fit(X = df[parameters], y = df[feature + '_imp'])
```

→ Stochastic Regression Imputation

```
predict = model.predict(df[parameters])  
std_error = (predict[df[feature].notnull()] - df.loc[df[feature].notnull(), feature + '_imp']).std()
```

→ amati bahwa kita menyimpan indeks data yang hilang dari kerangka data asli

```
random_predict = np.random.normal(size = df[feature].shape[0],  
                                  loc = predict,  
                                  scale = std_error)  
random_data.loc[(df[feature].isnull()) & (random_predict > 0), "Ran" + feature] = random_predict[(df[feature].isnull()) & (random_predict > 0)]
```

→ Kesalahan Standar dari perkiraan regresi sama dengan std() dari kesalahan setiap perkiraan

Hands On (Lanjutan)

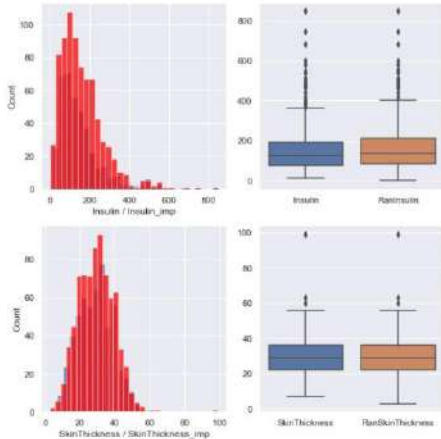
```
sns.set()
fig, axes = plt.subplots(nrows = 2, ncols = 2)
fig.set_size_inches(8, 8)

for index, variable in enumerate(["Insulin", "SkinThickness"]):
    sns.histplot(df[variable].dropna(), kde = False, ax = axes[index, 0])
    sns.histplot(random_data["Ran" + variable], kde = False, ax = axes[index, 0], color = 'red')
    axes[index, 0].set(xlabel = variable + " / " + variable + '_imp')

    sns.boxplot(data = pd.concat([df[variable], random_data["Ran" + variable]], axis = 1),
                ax = axes[index, 1])

plt.tight_layout()
```

→ membuat chart



→ Output

Hands On (Lanjutan)

```
pd.concat([df[["Insulin", "SkinThickness"]], random_data[["RanInsulin", "RanSkinThickness"]]], axis = 1).describe().T
```

	count	mean	std	min	25%	50%	75%	max
Insulin	394.0	155.548223	118.775855	14.000000	76.250000	125.000000	190.000000	846.0
SkinThickness	541.0	29.153420	10.476982	7.000000	22.000000	29.000000	36.000000	99.0
RanInsulin	768.0	159.817877	107.962192	1.350895	82.852612	137.134053	212.448998	846.0
RanSkinThickness	768.0	29.056917	10.211228	2.984951	22.000000	29.000000	36.000000	99.0

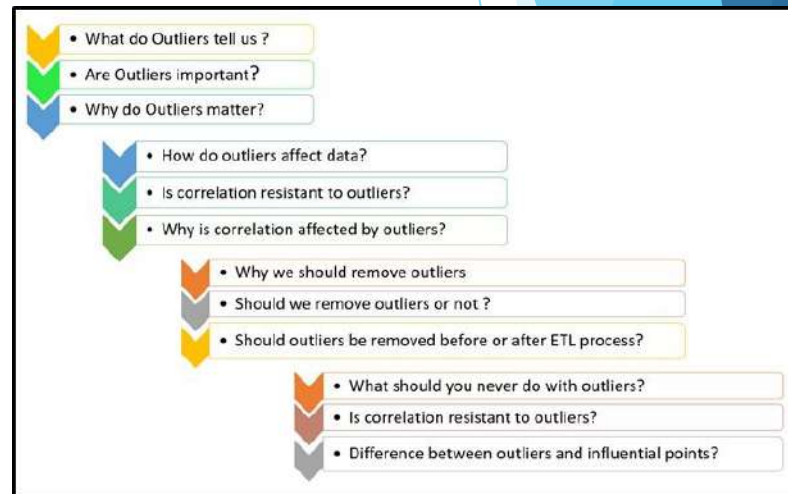
→ Output

Kesimpulan Teknik Imputasi

- Dua kategori imputasi:
 - Berdasarkan layer/tahapan: single imputation dan multiple imputation
- Tidak ada cara sempurna untuk mengkompensasi missing value (nilai yang hilang) dalam kumpulan data.
- Setiap strategi memiliki kinerja lebih baik untuk kumpulan data tertentu dan tipe data yang hilang tetapi dapat berkinerja jauh lebih buruk pada jenis kumpulan data lainnya.

Mengatasi (*Handling*) *Outlier* (HO)

- Definisi *Outlier*:
 - Titik data yang sangat berbeda dari data lainnya.
 - Pengamatan yang menyimpang dari pola keseluruhan pada sampel.
- Penyebab:
 - Error percobaan, salah input, error instrumen, kesengajaan (untuk pengujian), error pemrosesan data, error sampling, kewajaran karena keanehan dalam data (bukan error).



Mengatasi (*Handling*) *Outlier* (HO)

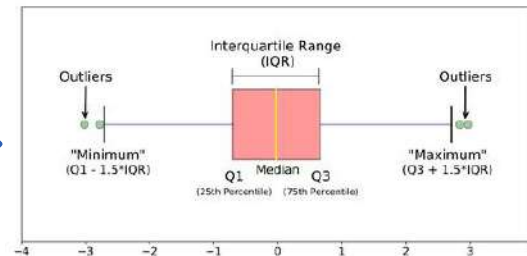
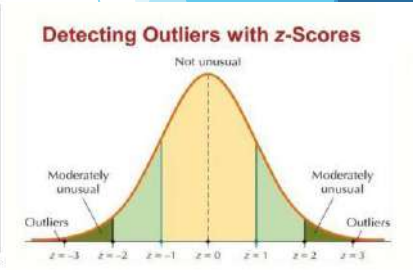
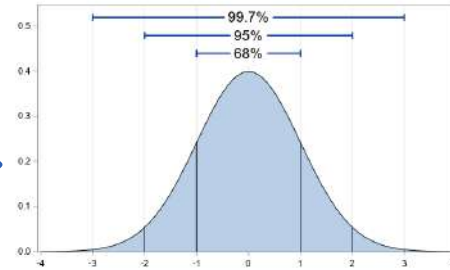
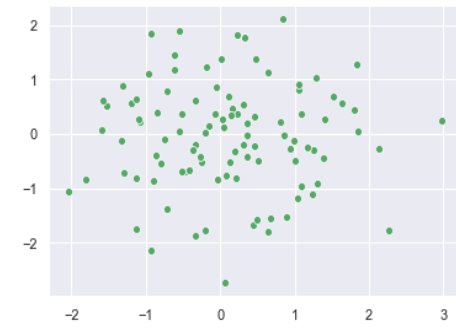
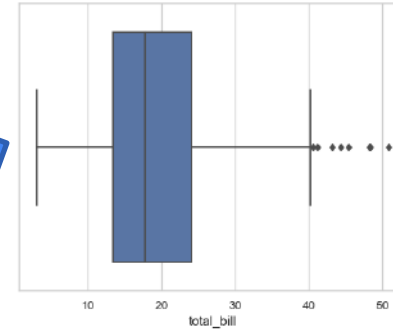
- Jenis/kategori:
 - Univariate vs multivariate
 - Parametrik vs non-parametrik
- Deteksi dan Cari Outlier dengan:
 - Visualisasi
 - Distribusi normal

Teknik Mengatasi Outlier:

- Trimming
- Winsorizing
- Imputing
- Discretization
- Censoring
- Z-score
- Linear Regression Model

Deteksi Outlier

- Visualisasi dgn Boxplot dan Scatterplot
 - Sebagian besar titik data terletak di tengah, tetapi ada satu titik yang jauh dari pengamatan lainnya; ini bisa menjadi outlier.
- Distribusi Normal
 - Dalam distribusi normal, sekitar 99,7% data berada dalam tiga standar deviasi dari mean.
 - Jika ada pengamatan yang lebih dari tiga kali standar deviasi, kemungkinan itu adalah outlier.
- Z-score
- Inter Quantile Range (IQR)



sum ber gbr
<https://heartbeat.fritz.ai/hands-on-with-feature-engineering-techniques-dealing-with-outliers-fc3f57cb63b>
<https://www.analyticsvidhya.com/blog/2021/05/detecting-and-treating-outliers-treating-the-odd-one-out/>

Teknik HO: Trimming (Pangkas) vs Winsorizing

- Nama lain: Truncation (Potong)
- Definisi: Menghapus outlier dari dataset
- Perlu memutuskan metrik untuk menentukan outlier.

Definisi:
Mengganti outlier dari dataset dengan nilai persentil setiap ujung/batas atas dan bawah.

Trimming vs Winsorizing

- Contoh kasus: laporan jumlah pasien yang ditangani tiap dokter/bulan (di bawah 100 pasien), dengan 4% dilaporkan lebih dari 100 pasien.



How many patients do you manage per month with Condition Y?



How many patients do you manage per month with Condition Y?



range [5,100].
Outlier tidak dibuang, namun dimasukan ke dalam range terdekat. Sehingga nilai mean mengecil. Median dan N tidak berubah.

Membuang data yang berada di luar range. Nilai N berubah, mean mengecil, median masih dinilai yg sama.

Hands On

```
import numpy as np
from scipy.stats.mstats import winsorize
from scipy.stats.mstats import trima
```

import numpy dan scipy

```
a = np.array([10, 4, 9, 8, 5, 3, 7, 2, 1, 6])
```

Masukkan array berisi 1 - 10

```
wins = winsorize(a, limits=[0.1, 0.2])
wins
```

Winsorize akan mengganti 10% nilai terendah dan 20% nilai tinggi

```
masked_array(data=[8, 4, 8, 8, 5, 3, 7, 2, 2, 6],
              mask=False,
              fill_value=999999)
```

→ Output

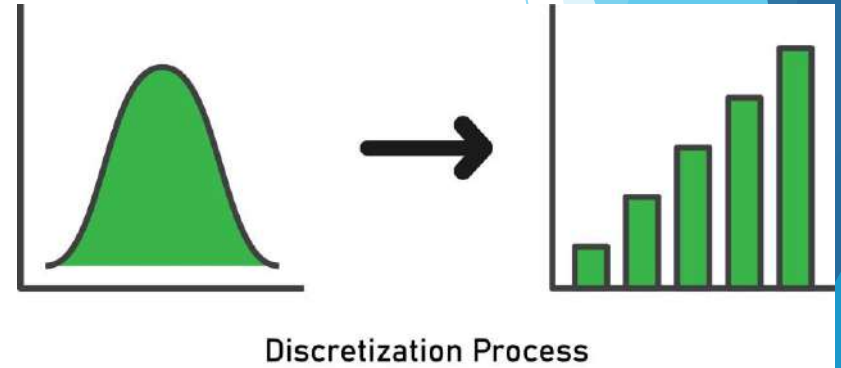
```
trims = trima(a, limits=(2,8))
print(trims)
```

Trims akan memotong nilai dengan batas tertentu

```
[-- 4 -- 8 5 3 7 2 -- 6] → Output
```

Discretization

- Definisi:
 - Proses mengubah fungsi, model dan variabel kontinu menjadi diskret (data kontinu di *Ukur* (measured) vs data kontinu di *Hitung* (counted)).
- Nama lain: Binning.
- Dasar Pertimbangan:
 - Data kontinu memiliki derajat kebebasan (DoF) yang tak hingga.
 - Data kontinu lebih mudah dipahami dan disimpan dalam bentuk kategori/grup
 - misal berat badan < 65 kg (ringan); 65 – 80 kg (mid); >80 kg (berat).

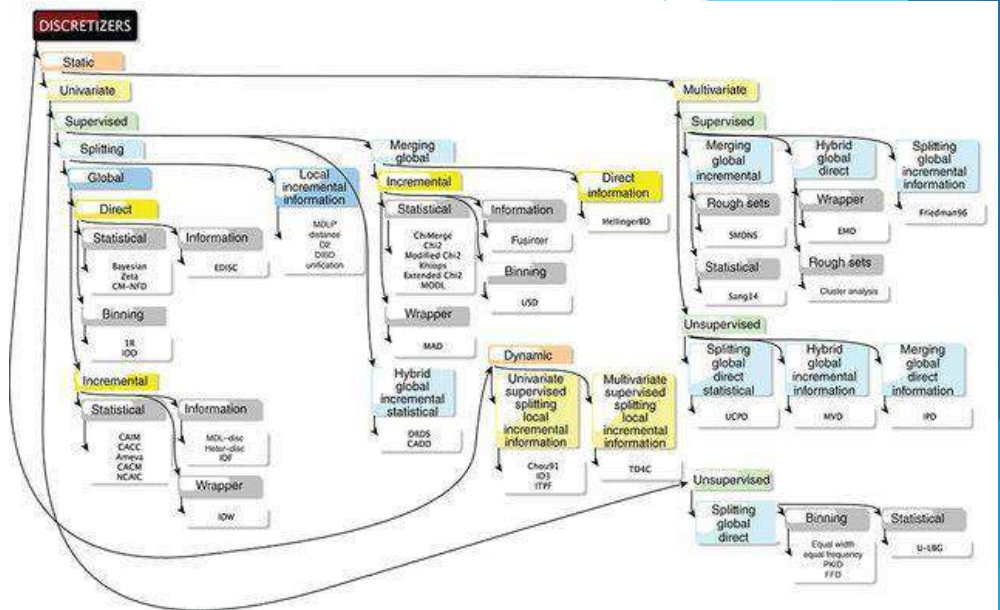


Discretization

- Jenis:

- Supervised
 - Decision Tree.
- Unsupervised
 - Equal-width discretization.
 - Equal-frequency discretization.
 - K-means discretization.
- Lainnya
 - Custom discretization.

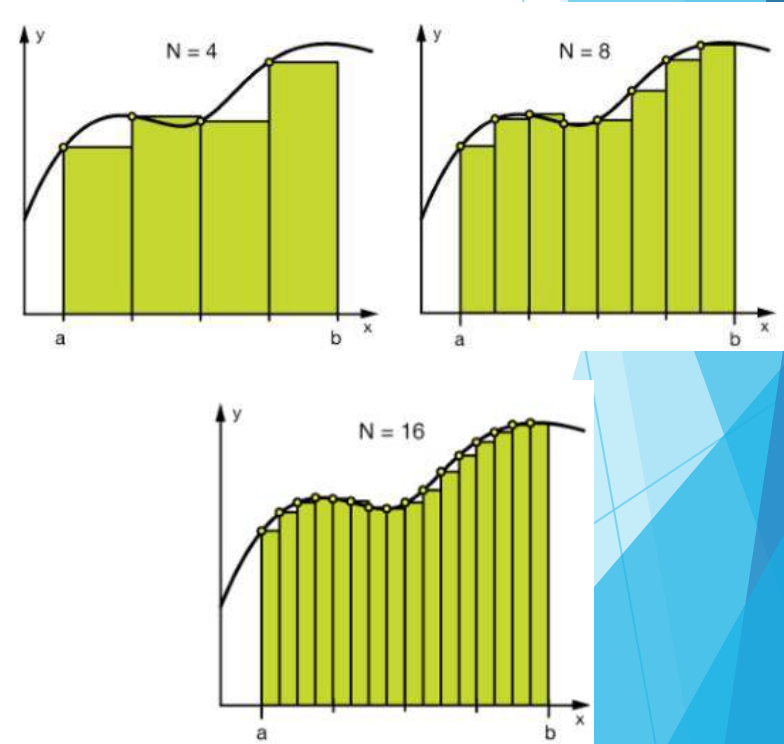
Taksonomi Discretization



Binning

- Pro:
 - Dapat diterapkan pada data kategorik dan numerik.
 - Model lebih robust dan mencegah overfitting.
- Kontra:
 - Meningkatnya biaya kinerja perhitungan.
 - Mengorbankan informasi.
 - Untuk kolom data numerik, dapat menyebabkan redundansi untuk beberapa algoritma.
 - Untuk kolom data kategorik, label dengan frekuensi rendah berdampak negatif pada robustness model statistik.
 - Untuk ukuran data dengan 100 ribu baris, disarankan menggabungkan label/kolom dengan record dengan < 100 menjadi kategori baru, misal "Lain-lain".

Ilustrasi binning untuk data numerik



Hands On

```
import pandas as pd
import numpy as np
```

```
df = pd.read_csv('bins.csv')
df
```

→ import pandas, numpy

	Item	Harga
0	Item_1	1500
1	Item_2	3300
2	Item_3	11000
3	Item_4	87500
4	Item_5	45000
5	Item_6	28600
6	Item_7	39900
7	Item_8	91000
8	Item_9	64700
9	Item_10	6000
10	Item_11	19000
11	Item_12	2700

→ Output

membuat array yang berisi sejumlah angka dengan jarak yang sama dengan linspace(), linspace --> numpy.linspace(nilai minimum, nilai maksimum, jumlah elemen)

```
bins = np.linspace(min(df['Harga']), max(df['Harga']), 4)
bins
```

```
array([ 1500.          , 31333.33333333, 61166.66666667, 91000.         ])
```

→ Output

```
df = pd.read_csv('bins.csv')
kategori = ['Murah', 'Standar', 'Mahal']
df['Harga Binned'] = pd.cut(df['Harga'], bins, labels=kategori, include_lowest=True)
df
```

	Item	Harga	Harga Binned
0	Item_1	1500	Murah
1	Item_2	3300	Murah
2	Item_3	11000	Murah
3	Item_4	87500	Mahal
4	Item_5	45000	Standar
5	Item_6	28600	Murah
6	Item_7	39900	Standar
7	Item_8	91000	Mahal
8	Item_9	64700	Mahal
9	Item_10	6000	Murah
10	Item_11	19000	Murah
11	Item_12	2700	Murah

→ Output

- bins --> mengelompokkan data
- binning dengan cut()
- membuat kategori harga

```
df = pd.read_csv('bins.csv')
interval_range = pd.interval_range(start=0, freq=10000, end=100000)
df['Harga Binned 2'] = pd.cut(df['Harga'], bins = interval_range)
df
```

	Item	Harga	Harga Binned 2
0	Item_1	1500	(0, 10000]
1	Item_2	3300	(0, 10000]
2	Item_3	11000	(10000, 20000]
3	Item_4	87500	(80000, 90000]
4	Item_5	45000	(40000, 50000]
5	Item_6	28600	(20000, 30000]
6	Item_7	39900	(30000, 40000]
7	Item_8	91000	(90000, 100000]
8	Item_9	64700	(60000, 70000]
9	Item_10	6000	(0, 10000]
10	Item_11	19000	(10000, 20000]
11	Item_12	2700	(0, 10000]

→ Output

menggunakan interval range untuk binning data dengan cut()

Hands On (Lanjutan)

```
df = pd.read_csv('bins.csv')
df['Harga Binned 3'] = pd.qcut(df['Harga'], 3)
df
```

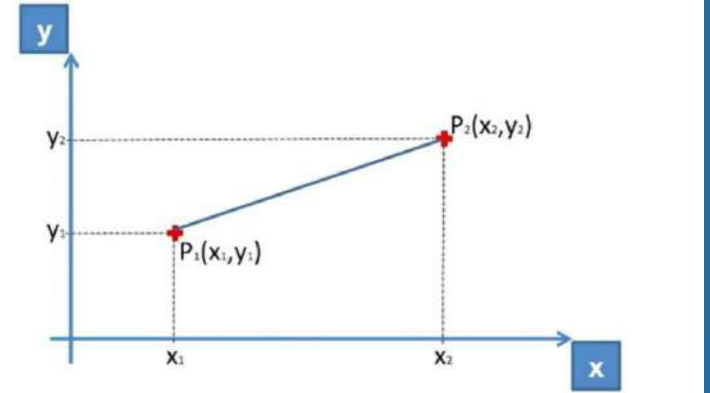
→ binning dengan qcut()

	Item	Harga	Harga Binned 3
0	Item_1	1500	(1499.999, 9333.333]
1	Item_2	3300	(1499.999, 9333.333]
2	Item_3	11000	(9333.333, 41600.0]
3	Item_4	87500	(41600.0, 91000.0]
4	Item_5	45000	(41600.0, 91000.0]
5	Item_6	28600	(9333.333, 41600.0]
6	Item_7	39900	(9333.333, 41600.0]
7	Item_8	91000	(41600.0, 91000.0]
8	Item_9	64700	(41600.0, 91000.0]
9	Item_10	6000	(1499.999, 9333.333]
10	Item_11	19000	(9333.333, 41600.0]
11	Item_12	2700	(1499.999, 9333.333]

→ Output

Scaling (Penskalaan)

- Dasar:
 - Sering diabaikan oleh pemula di Data Science.
 - Data numerik (biasanya) tidak memiliki range. range “Usia” vs range “Gaji” tidak sama (karakteristik berbeda). *Usia* memiliki rentang dari 1 sampai 150 (dalam tahun), sedangkan *Gaji* memiliki rentang dari 10 ribu sampai 100 ribu (dalam dolar). Untuk itu membandingkan perlu *scaling*.
 - Beberapa algoritma Machine Learning (regresi linear dan logistik dan Neural Network; SVM, KNN, K-means; LDA; PCA) yang menggunakan teknik optimasi Euclidian Distance 2 poin (titik).



$$\text{Euclidean Distance between } P_1 \text{ and } P_2 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- Dengan menggunakan rumus Euclidean Distance diatas, maka jelas bahwa hasil perhitungan pada kolom *Usia* dan *Gaji* akan memiliki jarak (distance) yang sangat jauh. Disinilah proses Feature Scaling dibutuhkan.
- Feature Scaling adalah suatu cara untuk membuat numerical data pada dataset memiliki rentang nilai (scale) yang sama. Tidak ada lagi satu variabel data yang mendominasi variabel data lainnya.

Hands On

```
import pandas as pd  
from sklearn.preprocessing import Normalizer
```

import library

```
data = pd.read_csv("scaling.csv")
```

membaca data

```
max_abs = Normalizer(norm = 'l2')
```

buat objek skalar dengan normalizer

```
max_abs.fit(data)
```

sesuaikan scaler dengan data

```
train_scaled = max_abs.transform(data)
```

mengubah data

```
train_scaled
```

```
array([[6.11110997e-04, 9.99999813e-01],  
       [5.62499911e-04, 9.9999842e-01],  
       [5.5555470e-04, 9.9999846e-01],  
       [6.22950699e-04, 9.9999806e-01],  
       [7.9999744e-04, 9.9999680e-01],  
       [6.03448166e-04, 9.9999818e-01],  
       [5.19230699e-04, 9.9999865e-01],  
       [6.07594825e-04, 9.9999815e-01],  
       [6.02409529e-04, 9.9999819e-01],  
       [5.52238722e-04, 9.9999848e-01]])
```

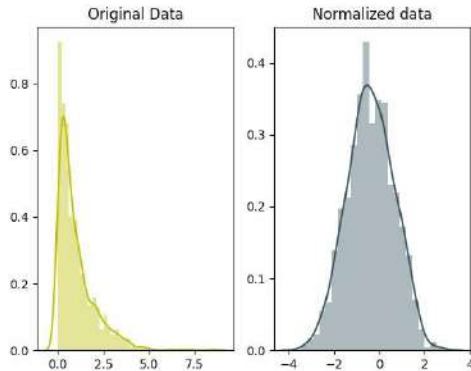
Output

Scaling: Jenis

Standardisation	Normalisation
$x_{\text{stand}} = \frac{x - \text{mean}(x)}{\text{standard deviation}(x)}$	$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$

Scaling: Normalisasi

- Nama Lain: Min-Max Scaling.
- Definisi: Teknik penskalaan di mana nilai-nilai digeser dan diubah skalanya sehingga nilainya berkisar antara 0 dan 1 (rentang $[0,1]$).



$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Di sini, X_{max} dan X_{min} masing-masing adalah nilai maksimum dan minimum dari fitur.

- Ketika nilai X adalah nilai minimum dalam kolom, pembilangnya adalah 0, dan karenanya X' adalah 0.
- Sebaliknya, ketika nilai X adalah nilai maksimum dalam kolom, pembilangnya sama dengan penyebutnya sehingga nilai X' adalah 1.
- Jika nilai X berada di antara nilai minimum dan maksimum, maka nilai X' berada di antara 0 dan 1.

Hands On

```
import pandas as pd
import numpy as np
```

import pandas
dan numpy

```
data = pd.read_csv("scalling.csv")
data
```

membaca data

	Age	Income
0	44	72000
1	27	48000
2	30	54000
3	38	61000
4	40	50000
5	35	58000
6	27	52000
7	48	79000
8	50	83000
9	37	67000

Output

```
means = data.mean(axis = 0)
```

menghitung mean

```
max_min = data.max(axis = 0) - data.min(axis = 0)
```

menghitung
max - min

```
train_scaled = (data - means) / max_min
```

menerapkan
transformasi ke
data

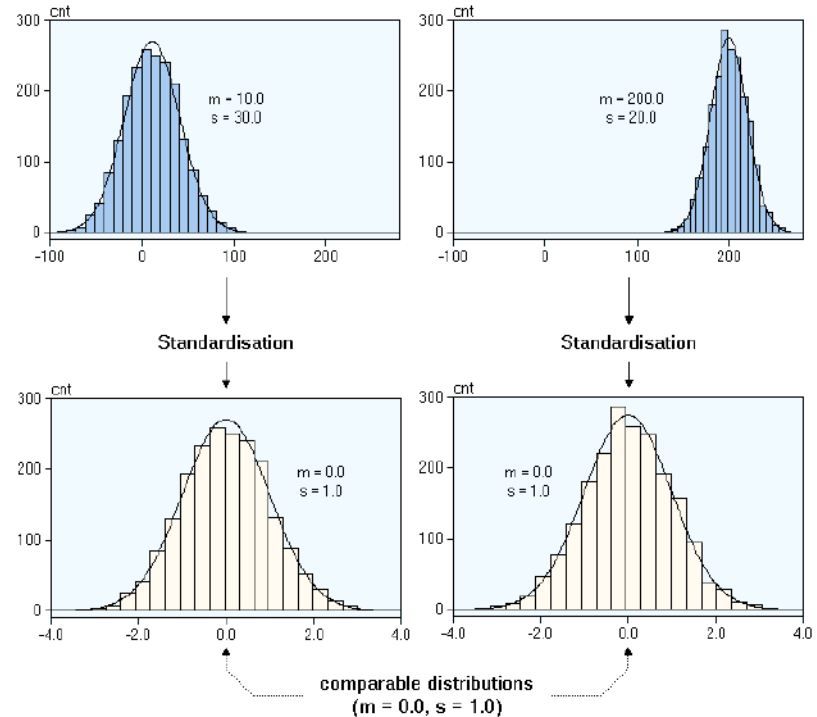
```
train_scaled
```

	Age	Income
0	0.278261	0.274286
1	-0.460870	-0.411429
2	-0.330435	-0.240000
3	0.017391	-0.040000
4	0.104348	-0.354286
5	-0.113043	-0.125714
6	-0.460870	-0.297143
7	0.452174	0.474286
8	0.539130	0.588571
9	-0.026087	0.131429

Output

Scaling: Standardisasi

- Tujuan: Berfokus pada mengubah data mentah menjadi informasi yang dapat digunakan sebelum dianalisis.
- Definisi: Teknik yang menskalakan data sehingga memiliki mean = 0 dan standar deviasi = 1
- Kontra:
 - Menambah langkah dalam data preparation
 - Waktu bertambah



Hands On

```
import pandas as pd  
from sklearn.preprocessing import StandardScaler
```

 → import library

```
data = pd.read_csv("scaling.csv")  
data
```

 → membaca data

	Age	Income
0	44	72000
1	27	48000
2	30	54000
3	38	61000
4	40	50000
5	35	58000
6	27	52000
7	48	79000
8	50	83000
9	37	67000

→ Output

```
scaler = StandardScaler()
```

 → buat objek scaler

```
scaler.fit(data)
```

 → sesuaikan scaler dengan data

```
train_scaled = scaler.transform(data)
```

 → mengubah data kereta dan uji

```
train_scaled
```

```
array([[ 0.8273403 ,  0.81886943],  
       [-1.37028238, -1.22830415],  
       [-0.98246661, -0.71651075],  
       [ 0.05170877, -0.11941846],  
       [ 0.31025261, -1.05770635],  
       [-0.336107  , -0.37531516],  
       [-1.37028238, -0.88710855],  
       [ 1.34442799,  1.41596173],  
       [ 1.60297184,  1.75715732],  
       [-0.07756315,  0.39237494]])
```

→ Output

Contoh Kasus *Scaling*

	Country	Age	Salary	Purchased
1	France	44	72000	No
2	Spain	27	48000	Yes
3	Germany	30	54000	No
4	Spain	38	61000	No
5	Germany	40		Yes
6	France	35	58000	Yes
7	Spain		52000	No
8	France	48	79000	Yes
9	Germany	50	83000	No
10	France	37	67000	Yes

The range of Age: 27 - 50

The range of Salary: 48,000 - 83,000

```
dataset['Age'].min()
```

```
27.0
```

```
dataset['Salary'].min()
```

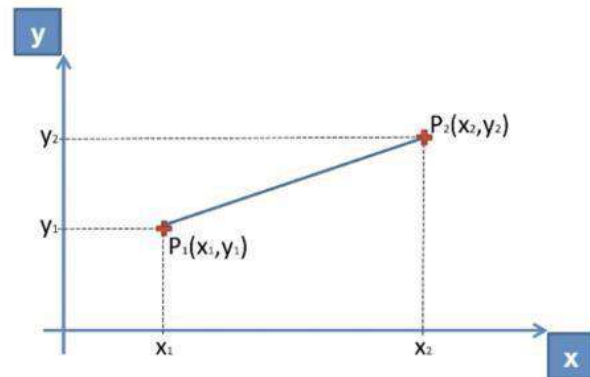
```
48000.0
```

```
dataset['Age'].max()
```

```
50.0
```

```
dataset['Salary'].max()
```

```
83000.0
```



$$\text{Euclidean Distance between } P_1 \text{ and } P_2 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Let x be the no. of Salary and y be the no. of Age

Example: x1&y1 are in row 2, x2&y2 are in row 9

$$(x_2 - x_1)^2 = (83000 - 48000)^2$$

$$= 1225000000$$

$$(y_2 - y_1)^2 = (50 - 27)^2$$

$$= 529$$

Contoh Kasus *Scaling*

- Ketika kita menghitung persamaan jarak (distance) Euclidean, jumlah $(x_2-x_1)^2$ jauh lebih besar daripada jumlah $(y_2-y_1)^2$ yang berarti jarak Euclidean akan didominasi oleh *Gaji* jika kita tidak menerapkan penskalaan. Perbedaan *Usia* berkontribusi lebih sedikit terhadap perbedaan keseluruhan.
- Oleh karena itu, kita harus menggunakan penskalaan untuk membawa semua nilai ke besaran yang sama dan dengan demikian, menyelesaikan masalah ini.

Standardisation

	Age	Salary
0	0.758874	7.494733e-01
1	-1.711504	-1.438178e+00
2	-1.275555	-8.912655e-01
3	-0.113024	-2.532004e-01
4	0.177609	6.632192e-16
5	-0.548973	-5.266569e-01
6	0.000000	-1.073570e+00
7	1.340140	1.387538e+00
8	1.630773	1.752147e+00
9	-0.258340	2.937125e-01

Max-Min Normalization

	Age	Salary
0	0.739130	0.685714
1	0.000000	0.000000
2	0.130435	0.171429
3	0.478261	0.371429
4	0.565217	0.450794
5	0.347826	0.285714
6	0.512077	0.114286
7	0.913043	0.885714
8	1.000000	1.000000
9	0.434783	0.542857

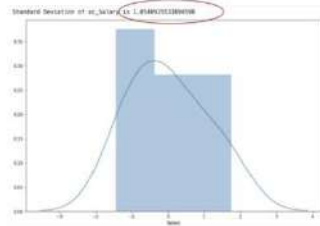
Contoh Kasus *Scaling*

After Feature scaling.

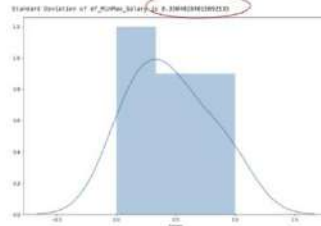
Column: Salary

Standard Deviation (Salary):
Max-Min Normalization (0.33) < Standardisation (1.05)

Standardisation



Max-Min Normalisation

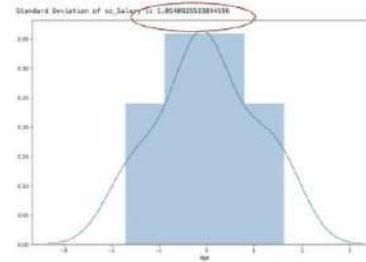


Normal distribution and Standard Deviation of Salary.

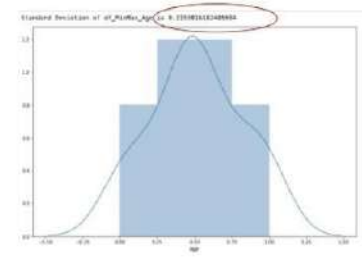
Column: Age

Standard Deviation (Age):
Max-Min Normalization (0.315) < Standardisation (1.05)

Standardisation



Max-Min Normalisation



Normal distribution and Standard Deviation of Age.

Referensi

- https://www.ud.ac.uk/population-health-sciences/sites/population-health-sciences/files/quartagno_1.pdf
- https://rianneschouten.github.io/missing_data_science/assets/blogpost/blogpost.html
- <https://towardsdatascience.com/tf-term-frequency-idf-inverse-document-frequency-from-scratch-in-python-6c2b61b78558>
- <https://dataaspirant.com/nlp-text-preprocessing-techniques-implementation-python>
- http://www.oreilly.com/library/view/blueprints-for-text/9781492074076/assets/btap_0401.png
- <https://monkeylearn.com/unstructured-data>
- <https://medium.com/machine-learning-id/melakukan-feature-scaling-pada-dataset-229531bb08de>
- <https://protobi.com/post/extreme-values-winsorize-trim-or-retain>
- <https://heartbeat.fritz.ai/hands-on-with-feature-engineering-techniques-dealing-with-outliers-fcc9f57cb63b>
- <https://www.analyticsvidhya.com/blog/2021/05/detecting-and-treating-outliers-treating-the-odd-one-out/>

Tools /Lab Online

- Jupyter Notebook
- Google Collabs

Summary

- Transformasi Data adalah bagian dari Data Preparation
- Membutuhkan pengetahuan dasar dan detail serta waktu yang mayoritas untuk menjamin data yang akan dianalisis sebersih mungkin
- Transformasi data dapat menggunakan beberapa teknik rekayasa fitur (feature engineering)
- Normalisasi, Standardisasi adalah bagian proses atau tahapan yang diperlukan untuk mentransformasi data
- Selain data terstruktur, transformasi data juga krusial dilakukan untuk data yang semi terstruktur dan tidak terstruktur (unstructured) seperti teks, image, audio dan video

TERIMA KASIH