

Laporan
Tugas Supervised Learning Kecerdasan Buatan
Kelas Kecerdasan Buatan D
2023/2024



Dibuat Oleh :

Krisna Bimantoro

(202210370311254)

PROGRAM STUDI INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS MUHAMMADIYAH MALANG
TAHUN 2024

Tugas Supervised Learning

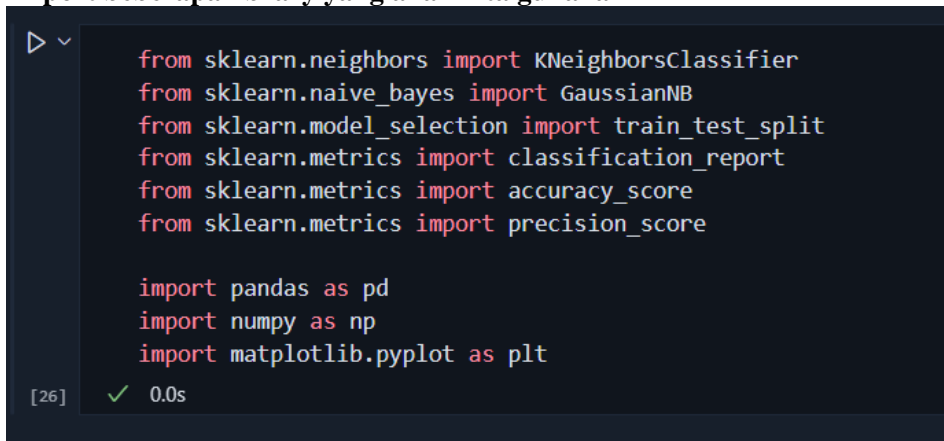
Kecerdasan Buatan 2023/2024

A. Jupyter Lab

Beberapa library yang harus diinstal dan disiapkan

- a. Sklearn (pip install scikit-learn)
- b. Pandas
- c. Numpy
- d. Matplotlib

1. Import beberapa library yang akan kita gunakan



```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[26] ✓ 0.0s

Pada import yang dilakukan pada library diatas digunakan untuk melakukan analysis data terhadap dataset. Diantaranya terdapat library

1. KNN: digunakan untuk penggunaan function pada algoritma KNN
2. Naive Bayes: digunakan untuk penggunaan function pada algoritma Naive Bayes
3. Train Test Split: library dari scikit learn yang digunakan untuk split data menjadi train dan test
4. Classification Report: report hasil akhir klasifikasi
5. Accuracy Score: menghitung akurasi penggunaan algoritma dalam dataset
6. Precision Score: menghitung presisi penggunaan algoritma dalam dataset
7. Pandas: digunakan untuk pembuatan dataframe untuk data visualisasi
8. Numpy: digunakan untuk akses array
9. Matplotlib.pyplot: digunakan untuk visualisasi data aktual dengan data klasifikasi

2. Download dataset dan import kedalam code

Dataset yang kita gunakan adalah dataset hepatitis dari <https://archive.ics.uci.edu/dataset/46/hepatitis>

```

from ucimlrepo import fetch_ucirepo

# fetch dataset
hepatitis = fetch_ucirepo(id=46)

# data (as pandas dataframes)
X = hepatitis.data.features
y = hepatitis.data.targets

# metadata
print(hepatitis.metadata)

# variable information
print(hepatitis.variables)

```

[27] ✓ 1.6s

Dari ucimlrepo import fetch_repo atau pengambilan dataset dengan menggunakan variabel hepatitis yang akan digunakan memiliki **id 46** pada ucimlrepo.

Menggunakan variabel X untuk feature dan y untuk target yang akan digunakan selanjutnya. Menampilkan metadata dari hepatitis dan seluruh variabel pada dataset sekaligus mengetahui variabel mana yang memiliki data kosong.

Output:

```

{'uci_id': 46, 'name': 'Hepatitis', 'repository_url': 'https://archive.ics.uci.edu/dataset/46/hepatitis', 'data_url':

```

	name	role	type	demographic	description	units	\
0	Class	Target	Categorical	None	None	None	
1	Age	Feature	Integer	None	None	None	
2	Sex	Feature	Categorical	None	None	None	
3	Steroid	Feature	Categorical	None	None	None	
4	Antivirals	Feature	Categorical	None	None	None	
5	Fatigue	Feature	Categorical	None	None	None	
6	Malaise	Feature	Categorical	None	None	None	
7	Anorexia	Feature	Categorical	None	None	None	
8	Liver Big	Feature	Categorical	None	None	None	
9	Liver Firm	Feature	Categorical	None	None	None	
10	Spleen Palpable	Feature	Categorical	None	None	None	
11	Spiders	Feature	Categorical	None	None	None	
12	Ascites	Feature	Categorical	None	None	None	
13	Varices	Feature	Categorical	None	None	None	
14	Bilirubin	Feature	Continuous	None	None	None	
15	Alk Phosphate	Feature	Integer	None	None	None	
16	Sgot	Feature	Integer	None	None	None	
17	Albumin	Feature	Integer	None	None	None	
18	Protine	Feature	Integer	None	None	None	
19	Histology	Feature	Integer	None	None	None	

	missing_values
0	no
1	no
2	no
3	yes
4	no
5	yes
6	yes
7	yes
8	yes
9	yes
10	yes
11	yes
12	yes
13	yes
14	yes
15	yes
16	yes
17	yes
18	yes
19	no

3. Preview dataset yang sudah kalian download

```
#code untuk preview fitur fitur yang digunakan (clue:.head())
print("Preview Fitur-fitur yang akan digunakan(X):")
print(X.head(10))

#code untuk preview target yang mau diklasifikasikan berdasarkan apa saja
print("Preview Target(y):")
print(y.head(10))

[28] ✓ 0.0s
```

Dengan menggunakan function .head() bisa menampilkan data yang dimulai dari head. Variabel X adalah feature yang telah di declare sebelumnya dan menampilkan 10 data. Variabel y adalah target yang telah di declare sebelumnya dan menampilkan 10 data.

Output:

Preview Fitur-fitur:									
	Age	Sex	Steroid	Antivirals	Fatigue	Malaise	Anorexia	Liver Big	\
0	30	2	1.0	2	2.0	2.0	2.0	1.0	
1	50	1	1.0	2	1.0	2.0	2.0	1.0	
2	78	1	2.0	2	1.0	2.0	2.0	2.0	
3	31	1	NaN	1	2.0	2.0	2.0	2.0	
4	34	1	2.0	2	2.0	2.0	2.0	2.0	
5	34	1	2.0	2	2.0	2.0	2.0	2.0	
6	51	1	1.0	2	1.0	2.0	1.0	2.0	
7	23	1	2.0	2	2.0	2.0	2.0	2.0	
8	39	1	2.0	2	1.0	2.0	2.0	2.0	
9	30	1	2.0	2	2.0	2.0	2.0	2.0	

	Liver Firm	Spleen Palpable	Spiders	Ascites	Varices	Bilirubin	\
0	2.0	2.0	2.0	2.0	2.0	1.0	
1	2.0	2.0	2.0	2.0	2.0	0.9	
2	2.0	2.0	2.0	2.0	2.0	0.7	
3	2.0	2.0	2.0	2.0	2.0	0.7	
4	2.0	2.0	2.0	2.0	2.0	1.0	
5	2.0	2.0	2.0	2.0	2.0	0.9	
6	2.0	1.0	1.0	2.0	2.0	NaN	
7	2.0	2.0	2.0	2.0	2.0	1.0	
8	1.0	2.0	2.0	2.0	2.0	0.7	
9	2.0	2.0	2.0	2.0	2.0	1.0	

	Alk Phosphate	Sgot	Albumin	Prottime	Histology
0	85.0	18.0	4.0	NaN	1
1	135.0	42.0	3.5	NaN	1
2	96.0	32.0	4.0	NaN	1
3	46.0	52.0	4.0	80.0	1
4	NaN	200.0	4.0	NaN	1
5	95.0	28.0	4.0	75.0	1
6	NaN	NaN	NaN	NaN	1
7	NaN	NaN	NaN	NaN	1
8	NaN	48.0	4.4	NaN	1
9	NaN	120.0	3.9	NaN	1

Pada output terdapat nilai null atau NaN pada beberapa index untuk beberapa variabel atau column. Maka perlu dilakukan cleaning data atau data preprocessing.

Preview Target:	
	Class
0	2
1	2
2	2
3	2
4	2
5	2
6	1
7	2
8	2
9	2

Target yang ditunjukkan menampilkan hasil **2** yang memiliki arti LIVE sertakan 1 untuk DIE.

4. Data Preprocessing

Data processing sangat penting sekali dilakukan karena untuk mencari dan menganalisa data apakah ada terdapat data yang belum bersih. Maksud dari belum bersih adalah terdapat data yang hilang, NaN, Null, dll. Sebelum di proses lebih lanjut data harus sudah bersih

```
#code untuk preprocessing (clue: (Data Cleaning) cek apakah terdapat data yang hilang, Nan, atau Null.
X = hepatitis.data.features
y = hepatitis.data.targets

# Cek apakah terdapat data yang hilang (NaN atau Null)
print("Cek data yang hilang pada fitur-fitur:")
print(X.isnull().sum())

print("\nCek data yang hilang pada target:")
print(y.isnull().sum())
```

Mendeklarasi kembali variabel X dan y sebagai features dan targets.

Menampilkan summary variabel mana saja yang memiliki nilai null menggunakan function `nama_variabel.isnull().sum()` serta dapat menampilkan berapa banyak yang memiliki nilai null pada features dan targets.

Output:

```
Cek data yang hilang pada fitur-fitur:
Age          0
Sex          0
Steroid      1
Antivirals   0
Fatigue      1
Malaise      1
Anorexia     1
Liver Big    10
Liver Firm   11
Spleen Palpable 5
Spiders      5
Ascites      5
Varices      5
Bilirubin    6
Alk Phosphate 29
Sgot         4
Albumin      16
Protime      67
Histology    0
dtype: int64
```

Pada variabel features(X) terdapat banyak variabel yang memiliki nilai null pada datanya. Maka dari itu perlu dilakukan cleaning dengan menghapus by row ataupun by column

```
Cek data yang hilang pada target:
Class       0
dtype: int64
```

Pada target(y) tidak terdapat data yang null

Menganalisa drop data by row dan column jika ada data yang null/NaN

Kedua metode tersebut menggunakan function *nama_variabel.dropna(axis=0/1)*

```
#by row
print(f"No. of drop Row X: {X.shape[0]}")
print(f"No. of drop Row Y: {y.shape[0]}")

x_drop = X.dropna(axis=0)
y_drop = y.loc[X.dropna(axis=0).index]

print("\nSetelah penghapusan data yang hilang by row:")
print(x_drop.isnull().sum())

print("\nPreview Fitur-fitur:")
print(x_drop.head(5))
print(y_drop.head(5))

print(f"No. of drop Row X: {x_drop.shape[0]}")
print(f"No. of drop Row Y: {y_drop.shape[0]}")
```

By Row

Untuk by row menggunakan axis=0 yang digunakan pada variabel X atau features yang artinya jika satu baris pada variabel X memiliki nilai null/NaN pada satu variabel atau lebih maka baris tersebut akan di drop atau di hapus dan nilai yang tidak memiliki nilai null akan ditempatkan di variabel x_drop.

Pada variabel y_drop digunakan agar nilai/jumlah dari baris serta index yang ada pada variabel targets sama dengan x_drop hal ini sangat penting dilakukan agar variabel X dan y memiliki nilai yang constant atau sama.

Menggunakan function shape[0] yang digunakan untuk menampilkan jumlah baris. Jika ingin menampilkan jumlah kolom bisa menggunakan shape[1]

Output:

```
No. of drop Row X: 155
No. of drop Row Y: 155
```

Sebelum data di drop by row jumlah baris/data dari kedua variabel adalah 155

```
Setelah penghapusan data yang hilang by row:
Age          0
Sex          0
Steroid      0
Antivirals   0
Fatigue      0
Malaise      0
Anorexia     0
Liver Big    0
Liver Firm   0
Spleen Palpable 0
Spiders      0
Ascites      0
Varices      0
Bilirubin    0
Alk Phosphate 0
Sgot         0
Albumin      0
Protime      0
Histology    0
dtype: int64
```

Setelah melakukan drop by row setiap variabel X atau features sudah tidak memiliki nilai null lagi.

Preview Fitur-fitur:

	Age	Sex	Steroid	Antivirals	Fatigue	Malaise	Anorexia	Liver Big	\
5	34	1	2.0	2	2.0	2.0	2.0	2.0	
10	39	1	1.0	1	2.0	2.0	2.0	2.0	1.0
11	32	1	2.0	1	1.0	2.0	2.0	2.0	2.0
12	41	1	2.0	1	1.0	2.0	2.0	2.0	2.0
13	30	1	2.0	2	1.0	2.0	2.0	2.0	2.0

	Liver Firm	Spleen Palpable	Spiders	Ascites	Varices	Billirubin	\
5	2.0	2.0	2.0	2.0	2.0	0.9	
10	1.0	2.0	2.0	2.0	2.0	1.3	
11	1.0	2.0	1.0	2.0	2.0	1.0	
12	1.0	2.0	2.0	2.0	2.0	0.9	
13	1.0	2.0	2.0	2.0	2.0	2.2	

	Alk Phosphate	Sgot	Albumin	Prottime	Histology
5	95.0	28.0	4.0	75.0	1
10	78.0	30.0	4.4	85.0	1
11	59.0	249.0	3.7	54.0	1
12	81.0	60.0	3.9	52.0	1
13	57.0	144.0	4.9	78.0	1

Dan output dari variabel feature menunjukkan bahwa terdapat penghapusan baris yang di buktikan dengan adanya hilangnya index 1 sampai 4 dan beberapa index lainnya. Karena pada index tersebut terdapat data null/NaN. Output ini menggunakan function `x_drop.head(5)`

	Class
5	2
10	2
11	2
12	2
13	2

Pada targets atau variabel y menunjukkan adanya kesamaan mulainya data dari index yang sama dengan `x_drop`. Menggunakan `y_drop = y.loc[X.dropna(axis=0).index]`

```
No. of Row X: 80
No. of Row Y: 80
```

Setelah melakukan drop number of row atau banyaknya data otomatis berubah/berkurang dikarenakan terdapat beberapa data yang dihilangkan. Data sekarang sebanyak 80 baris.

By Column

```
#by col
x_drop_col = X.dropna(axis=1)
print("\n")
print(x_drop_col.isnull().sum())
print("\nPreview Fitur-fitur:")
print(x_drop_col.head(10))
```

Untuk by row menggunakan `axis=1` yang digunakan pada variabel X atau features yang artinya jika satu kolom pada variabel X memiliki nilai null/NaN maka kolom tersebut akan di drop atau di hapus dan nilai yang tidak memiliki nilai null akan ditempatkan di variabel `x_drop_col`.

Output:

Preview Fitur-fitur:				
	Age	Sex	Antivirals	Histology
0	30	2	2	1
1	50	1	2	1
2	78	1	2	1
3	31	1	1	1
4	34	1	2	1
5	34	1	2	1
6	51	1	2	1
7	23	1	2	1
8	39	1	2	1
9	30	1	2	1

Terlihat bahwa variabel feature sekarang menjadi sedikit atau hanya 4. Hal ini dikarenakan variabel yang lain memiliki nilai null/NaN sehingga di hapus dari variabel features sehingga hasil akhirnya maka **urutan data sesuai index tetap aman** yang artinya tidak mengalami penghapusan baris.

Note:

Pengujian berikutnya akan menggunakan by row dikarenakan dibutuhkannya banyak variabel X untuk membuktikan apakah memiliki pengaruh terhadap variabel y untuk memperkuat validitas hasil klasifikasi y.

5. Lakukan Split data menjadi train dan test

Split data merupakan tahap penting dalam supervised learning. Karena algoritma nanti akan mempelajari data pada data train dan menguji hasil belajarnya pada data test.

```
print(f"No. of X: {x_drop.shape[0]}")  
print(f"No. of Y: {y_drop.shape[0]}")
```

Melakukan pengecekan dahulu untuk mengetahui berapa jumlah data pada variabel x dan y dengan function `.shape[0]`.

Output:

```
No. of X: 80  
No. of Y: 80
```

Splitting

```
#Code untuk splitting dataset  
x_train, x_test, y_train, y_test = train_test_split(x_drop, y_drop, test_size=0.2, random_state=25, shuffle=False)
```

Hasil dari splitting ini lah yang nantinya akan digunakan untuk di train dan di test datanya pada algoritma kemudian akan menghasilkan 2 jenis variabel berbeda yaitu train dan test.

Pada code splitting menggunakan 4 variabel diantaranya:

`x_train`: Variabel ini digunakan untuk data yang akan di train pada feature

`x_test`: Variabel ini digunakan untuk data yang akan di test pada feature

`y_train`: Variabel ini digunakan untuk data yang akan di train pada target

`y_test`: Variabel ini digunakan untuk data yang akan di test pada target

Penjelasan

Kemudian menggunakan **function** `train_test_split()`

Function ini berfungsi untuk membagi dari data array menjadi train dan test. Terdapat beberapa **parameter** pada function ini.

X_drop dan **y_drop** adalah array yang akan digunakan, `X_drop` digunakan untuk `x_train` dan `x_test` sedangkan `y_drop` digunakan untuk `y_train` dan `y_test`.

Test_size merupakan besaran dari variabel test, nilai dari test size sendiri berkisar antara 0.0 sampai 1.0 pada kali ini menggunakan test size dengan besar 0.2. Sehingga hasil akhirnya nanti banyak data pada test size sebanyak 20% dari data mentah.

Random state digunakan untuk meng-shuffle data sebelum menerapkan split.

Shuffle digunakan agar hasil akhir datanya nanti tidak terjadi pengacakan data, sehingga masih memiliki index yang urut seperti awal tadi

```
# print(training_data.head(10))

print(f"No. of training X: {X_train.shape[0]}")
print(f"No. of testing X: {X_test.shape[0]}")

print(f"No. of training Y: {y_train.shape[0]}")
print(f"No. of testing Y: {y_test.shape[0]}")
```

[30] ✓ 0.0s

Menggunakan function `nama_variabel.shape[0]` untuk melihat jumlah data pada variabel train dan tes.

Output:

```
No. of training X: 64
No. of testing X: 16
No. of training Y: 64
No. of testing Y: 16
```

Hasil output menjelaskan bahwa setelah splitting data pada variabel training memiliki nilai 64 nilai ini didapat dari sisa data yang telah di ambil untuk variabel testing atau sebanyak 64 data, variabel ini lah yang nantinya digunakan untuk di training

Pada variabel testing memiliki nilai 16 yang didapat dari 20% dari jumlah data awal, 20% dari 80 adalah 16. Variabel ini digunakan untuk melakukan pengujian pada algoritma

6. Proses Learning dengan Algoritma Supervised Learning

Pada tugas ini kita akan menggunakan 2 algoritma

1. KNN
2. Naive Bayes

a. Algoritma KNN

```
▶ ▾  
# code panggil fungsi KNN pada library sklearn  
neigh = KNeighborsClassifier(n_neighbors=4)  
  
# code masukkan datatrain pada KNN dan jalankan proses learningnya  
neigh.fit(X_train, y_train)  
[31] ✓ 0.0s
```

Pada algoritma KNN menggunakan function *KNeighborsClassifier()* dengan parameter *n_neighbors=4* function inilah yang berfungsi untuk pemanggilan KNN pada library sklearn.

Parameter *n_neighbors=4* dapat diartikan untuk menentukan setiap data termasuk kedalam klasifikasi yang mana, ini tergantung dari kedekatan/ketetanggaan setiap data dengan ke 4 data disekitarnya.

Untuk melakukan data train pada KNN menggunakan function *neigh.fit(X_train, y_train)*. Pada parameter yang ada pada function ini menggunakan variabel x yang akan di train dan juga variabel y yang akan di train.

Ouptut:

```
▼ KNeighborsClassifier  
KNeighborsClassifier(n_neighbors=4)
```

Uji hasil Learning KNN

```
# code untuk pengujian hasil learning menggunakan KNN pada Datatest  
predicted = neigh.predict(X_test)  
  
# code hasil output  
print("Nilai Data Aktual")  
print(y_test)  
  
print("Nilai Data Hasil Klasifikasi")  
print(predicted)
```

Untuk pengujiannya dengan menggunakan variabel tambahan bernama *predictec* untuk menampung hasil uji. Dengan menggunakan function *neigh.predict()* untuk menguji hasil data yang di train, menggunakan variabel pengujian *X_test* yang dimasukkan kedalam parameter function untuk mendapatkan data hasil klasifikasi.

Hasil uji ini yang akan digunakan sebagai hasil klasifikasi

Output:

```

Nilai Data Aktual
Class
124    2
125    2
127    1
128    2
129    1
130    2
133    2
134    1
135    2
137    2
138    1
139    2
143    1
145    2
153    2
154    1
Nilai Data Hasil Klasifikasi
[2 2 2 2 2 1 2 2 2 1 2 2 2 2 1 1]

```

Pada output dapat dibandingkan data aktualnya (y_test) dengan hasil klasifikasi menunjukan perbedaan nilai pada beberapa index. Ini menunjukkan bahwa algoritma KNN bisa mengklasifikasi ulang untuk mendapatkan keakuratan yang lebih dengan menggunakan pada data terhadap data disekitarnya(neighbors).

```

# code visualisasi hasil output dengan membandingkan data aktual dengan data hasil klasifikasi
comparison_df = pd.DataFrame({
    'Aktual': y_test['Class'].values.flatten(),
    'Prediksi': predicted.flatten()
})

# Menghitung jumlah aktual dan prediksi
aktual_counts = comparison_df['Aktual'].value_counts().sort_index()
prediksi_counts = comparison_df['Prediksi'].value_counts().sort_index()

# Menyusun data untuk plotting
labels = sorted(set(comparison_df['Aktual']))
bar_width = 0.35
index = np.arange(len(labels))

# Membuat bar untuk data aktual
plt.figure(figsize=(12, 6))
plt.bar(index, [aktual_counts.get(label, 0) for label in labels], bar_width, label='Data Aktual', color='b', alpha=0.7)

# Membuat bar untuk prediksi
plt.bar(index + bar_width, [prediksi_counts.get(label, 0) for label in labels], bar_width, label='Hasil Klasifikasi (KNN)', color='r', alpha=0.7)

# Menambahkan label, judul, dan legenda
plt.xlabel('Kelas')
plt.ylabel('Jumlah')
plt.title('Perbandingan Data Aktual dan Hasil Klasifikasi (KNN)')
plt.xticks(index + bar_width / 2, labels)
plt.legend()

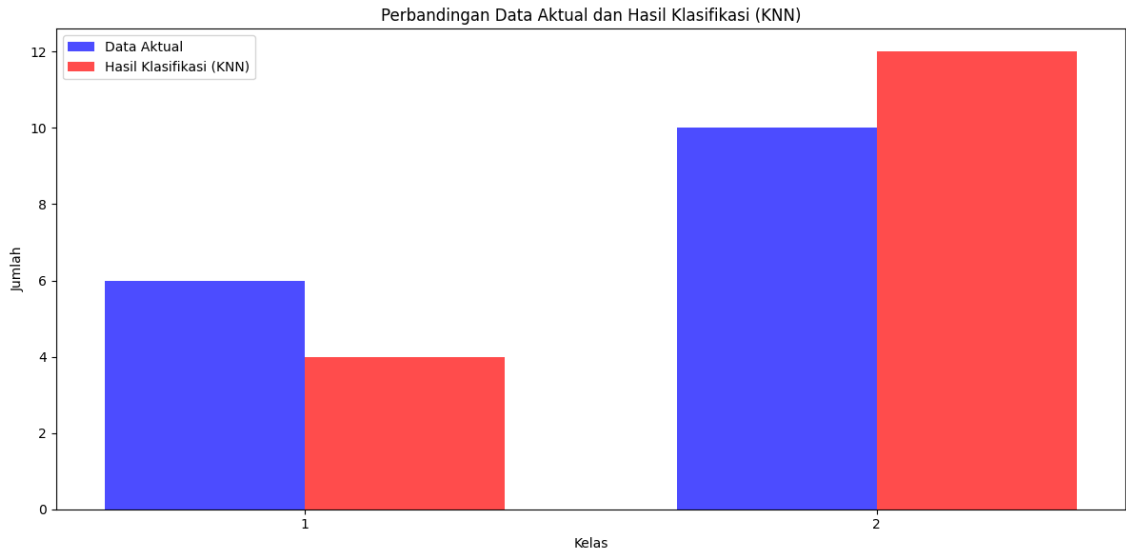
# Menampilkan plot
plt.tight_layout()
plt.show()

```

Untuk melihat perbandingan lebih lanjut menggunakan visualisasi graph batang dengan membandingkan antara data aktual dengan data hasil klasifikasi menggunakan matplotlib.

Dengan menggunakan pandas memasukkan kedua data kedalam data frame kemudian membuat variabel baru untuk menghitung nilai aktual dan prediksi. Kemudian di tampilkan dalam visualisasi

Output:



Pada hasil output dapat dilihat perbedaan jumlah pada data aktual dan hasil klasifikasi menggunakan KNN. Pada graph Nilai baris X adalah jumlah banyaknya data sedangkan baris Y menunjukkan klasifikasi nilai pada variabel target (Class) 1 atau 2. 1 untuk Die dan 2 untuk Live.

Graph menunjukan untuk nilai 1 pada data aktual lebih besar jumlahnya dibandingkan data klasifikasi. Namun sebaliknya pada nilai 2 lebih sering muncul pada hasil klasifikasi KNN dibandingkan data aktual.

b. Algoritma Naive Bayes

```
# code panggil fungsi Naive Bayes pada library sklearn
nb = GaussianNB()

# code masukkan datatrain pada KNN dan jalankan proses learningnya
nb.fit(X_train, y_train)
```

[33] ✓ 0.0s

Pada algoritma Naïve Bayes dengan menggunakan Gaussian menggunakan function *GaussianNB()* function inilah yang berfungsi untuk pemanggilan Naïve Bayes pada library sklearn.

Untuk melakukan data train pada KNN menggunakan function *nb.fit(X_train, y_train)*. Pada parameter yang ada pada function ini menggunakan variabel x yang akan di train dan juga variabel y yang akan di train.

```
▼ GaussianNB
GaussianNB()
```

Uji hasil Learning Naive Bayes

```

# code untuk pengujian hasil learning menggunakan Naive Bayes pada Datatest
y_pred = nb.predict(X_test)

# code hasil output
print("Nilai Data Aktual")
print(y_test)

print("Nilai Data Hasil Klasifikasi")
print(y_pred)

```

Untuk pengujiannya dengan menggunakan variabel tambahan bernama *y_pred* untuk menampung hasil uji. Dengan menggunakan function *nb.predict()* untuk menguji hasil data yang di train, menggunakan variabel pengujian *X_test* yang dimasukkan kedalam parameter function untuk mendapatkan data hasil klasifikasi.

Hasil uji ini yang akan digunakan sebagai hasil klasifikasi

Output:

```

Nilai Data Aktual
Class
124      2
125      2
127      1
128      2
129      1
130      2
133      2
134      1
135      2
137      2
138      1
139      2
143      1
145      2
153      2
154      1
Nilai Data Hasil Klasifikasi
[1 2 1 1 1 1 1 1 2 1 1 2 1 1 2 1]

```

Pada output dapat dibandingkan data aktualnya (*y_test*) dengan hasil klasifikasi menunjukan perbedaan nilai pada beberapa index. Hasil klasifikasi didapat dari variabel *y_pred*

```

# code visualisasi hasil output dengan membandingkan data aktual dengan data hasil klasifikasi
comparison_df = pd.DataFrame({
    'Aktual': y_test['Class'].values.flatten(),
    'Prediksi': y_pred.flatten()
})

# Menghitung jumlah aktual dan prediksi
aktual_counts = comparison_df['Aktual'].value_counts().sort_index()
prediksi_counts = comparison_df['Prediksi'].value_counts().sort_index()

# Menyusun data untuk plotting
labels = sorted(set(comparison_df['Aktual']))
bar_width = 0.35
index = np.arange(len(labels))

# Membuat bar untuk data aktual
plt.figure(figsize=(12, 6))
plt.bar(index, [aktual_counts.get(label, 0) for label in labels], bar_width, label='Data Aktual', color='b', alpha=0.7)

# Membuat bar untuk prediksi
plt.bar(index + bar_width, [prediksi_counts.get(label, 0) for label in labels], bar_width, label='Hasil Klasifikasi Naive Bayes', color='r', alpha=0.7)

# Menambahkan label, judul, dan legenda
plt.xlabel('Kelas')
plt.ylabel('Jumlah')
plt.title('Perbandingan Data Aktual dan Hasil Klasifikasi Naive Bayes')
plt.xticks(index + bar_width / 2, labels)
plt.legend()

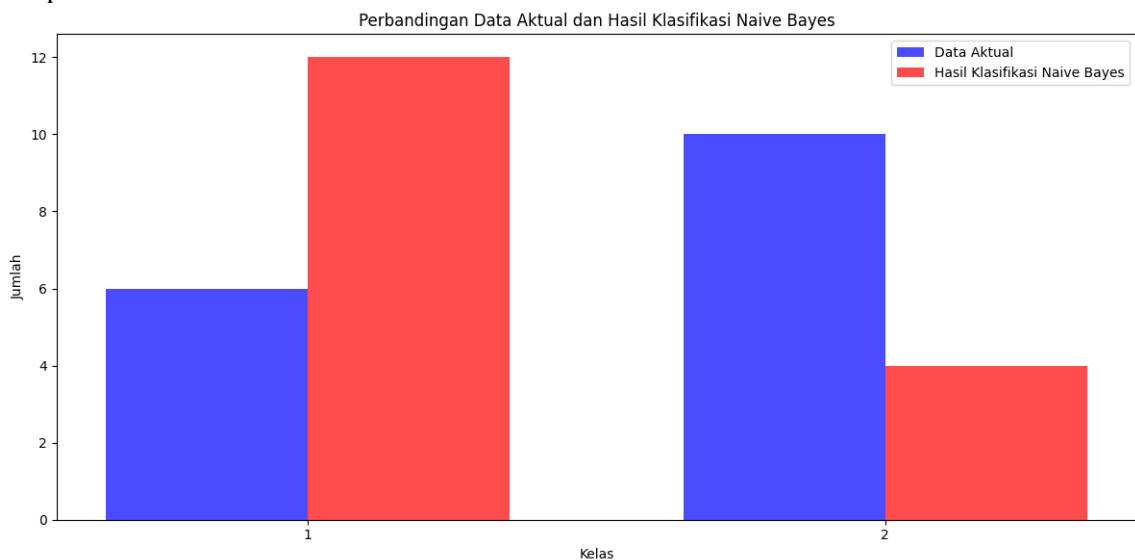
# Menampilkan plot
plt.tight_layout()
plt.show()

```

Untuk melihat perbandingan lebih lanjut menggunakan visualisasi graph batang dengan membandingkan antara data aktual dengan data hasil klasifikasi menggunakan matplotlib.

Dengan menggunakan pandas memasukkan kedua data kedalam data frame kemudian membuat variabel baru untuk menghitung nilai aktual dan prediksi. Kemudian di tampilkan dalam visualisasi

Output:



Pada hasil output dapat dilihat perbedaan jumlah pada data aktual dan hasil klasifikasi menggunakan Naïve Bayes. Pada graph Nilai baris X adalah jumlah banyaknya data sedangkan baris Y menunjukkan klasifikasi nilai pada variabel target (Class) 1 atau 2. 1 untuk Die dan 2 untuk Live.

Graph menunjukan untuk nilai 1 pada data aktual lebih kecil jumlahnya dibandingkan data klasifikasi. Namun sebaliknya pada nilai 2 lebih besar pada data aktual dibandingkan data hasil klasifikasi *

7. Pengujian

Setelah kita membuat model klasifikasi dengan menggunakan algoritma KNN dan Naive Bayes, hal selanjutnya yaitu kita perlu menguji algoritma mana yang paling baik di antara 2 algoritma tersebut. Metrics atau satuan pengukuran yang akan kita gunakan adalah Accuracy dan Precision.

a. Hitung nilai akurasi dan presisi dari model KNN

```
# code untuk menghitung akurasi
print("Nilai akurasi KNN sebesar ",accuracy_score(y_test, predicted))

# code untuk menghitung presisi
print("Nilai Presisi KNN sebesar ",precision_score(y_test, predicted))

print("\nHasil Classification")
print(classification_report(y_test, predicted))

[40] ✓ 0.0s
```

Accuracy

Untuk menghitung akurasi dari algoritma KNN menggunakan function yang suda ada pada library sklearn. Functoin yang digunakan yaitu *accuracy_score(data_aktual, data_prediksi)* dengan aturan kedu data harus memiliki jumlah yang sama.

Nilai akurasi yang bagus adalah mendekati 1 dan terburuk adalah mendekati 0.

Precision

Untuk menghitung presisi dari algoritma KNN menggunakan function yang suda ada pada library sklearn. Functoin yang digunakan yaitu *precision_score(data_aktual, data_prediksi)* dengan aturan kedu data harus memiliki jumlah yang sama.

Presisi adalah ratio $tp/(tp+fp)$ dimana tp adalah angka true positif dan fp adalah angka false positif. Presisi (precision) secara intuitif adalah kemampuan dari suatu classifier untuk tidak memberikan label positif pada sampel yang sebenarnya negatif.

Nilai presisi yang bagus adalah mendekati 1 dan terburuk adalah mendekati 0.

Classification Report

Function ini berguna untuk memberikan laporan klasifikasi mulai dari presisi, akurasi, recall, fi-score dan lain-lain

Output:


```

Nilai akurasi KNN sebesar 0.5
Nilai Presisi KNN sebesar 0.25

Hasil Classification
      precision    recall  f1-score   support

     1       0.25       0.17       0.20         6
     2       0.58       0.70       0.64        10

 accuracy          0.50         16
  macro avg       0.42       0.43       0.42         16
 weighted avg     0.46       0.50       0.47         16

```

Pada output menjelaskan bahwa algoritma KNN memiliki keakurasian sebesar 0.5 dan presisi sebesar 0.25.

b. Hitung nilai akurasi dan presisi dari model Naive Bayes

```

# code untuk menghitung akurasi
print("Nilai akurasi Naive Bayes sebesar ",accuracy_score(y_test, y_pred))

# code untuk menghitung presisi
print("Nilai Presisi Naive Bayes sebesar ",precision_score(y_test, y_pred))

print("\nHasil Classification")
print(classification_report(y_test, y_pred))
✓ 0.0s

```

Accuracy

Untuk menghitung akurasi dari algoritma Naïve Bayes menggunakan function yang suda ada pada library sklearn. Function yang digunakan yaitu *accuracy_score(data_aktual, data_prediksi)* dengan aturan kedu data harus memiliki jumlah yang sama.

Nilai akurasi yang bagus adalah mendekati 1 dan terburuk adalah mendekati 0.

Precision

Untuk menghitung presisi dari algoritma Naïve Bayes menggunakan function yang suda ada pada library sklearn. Functoin yang digunakan yaitu *precision_score(data_aktual, data_prediksi)* dengan aturan kedu data harus memiliki jumlah yang sama.

Presisi adalah ratio $tp/(tp+fp)$ dimana tp adalah angka true positif dan fp adalah angka false positif. Presisi (precision) secara intuitif adalah kemampuan dari suatu classifier untuk tidak memberikan label positif pada sampel yang sebenarnya negatif.

Nilai presisi yang bagus adalah mendekati 1 dan terburuk adalah mendekati 0.

Classification Report

Function ini berguna untuk memberikan laporan klasifikasi mulai dari presisi, akurasi, recall, fi-score dan lain-lain

Output:

Nilai akurasi Naive Bayes sebesar	0.625			
Nilai Presisi Naive Bayes sebesar	0.5			
	precision	recall	f1-score	support
1	0.50	1.00	0.67	6
2	1.00	0.40	0.57	10
accuracy			0.62	16
macro avg	0.75	0.70	0.62	16
weighted avg	0.81	0.62	0.61	16

Pada output menjelaskan bahwa algoritma Naïve Bayes memiliki keakurasian sebesar 0.625 dan presisi sebesar 0.5.

B. Kesimpulan

Dari apa yang telah kalian lakukan dapat disimpulkan dengan menjawab beberapa pertanyaan berikut:

1. Mengapa dataset perlu dilakukan tahap preprocessing?

Tahap ini sangat diperlukan untuk melakukan cleaning data, cleaning data disini dilakukan agar data mentah yang ingin digunakan tidak terdapat data yang kosong pada baris ataupun kolom yang nantinya akan digunakan sebagai data training dan data test.

Apabila terdapat data yang kosong pada salah satu data akan mengakibatkan beberapa hal diantaranya ialah: kesalahan perhitungan, ketidak akuratan hasil, kinerja algoritma menurun dan probabilitas tidak akurat pada algooritma naïve bayes

Namun data preprocessing disini tidak hanya menghilangkan data yang kosong, namun masih terdapat beberapa tahap lagi agar data tersebut layak digunakan untuk proses learning. Diantaranya ialah: Menghapus duplikasi data, menangani data yang tidak relevan (outlier) dan data yang tidak normal.

Pada analisa kali ini hanya menggunakan data preprocessing untuk menangani data yang kosong. Dari data yang sudah dilakukan preprocessing menghasilka bahwa terdapat **75** data kosong di variabel feature yang berbeda, maka dari itu dilakukan drop by row sehingga jumlah data menjadi **80**.

2. Mengapa dataset perlu dilakukan splitting?

Hasil splitting data akan menghasilkan 2 set yaitu set training yaitu test dan training. Kedua set inilah yang nantinya akan digunakan untuk learning dan uji pada algoritma KNN maupun Naïve Bayes. Karena jika tidak di split datanya kita tidak akan bisa melakukan training maupun uji.

Karena dibutuhkannya data aktual (data yang tidak disentuh sama sekali pada learning) untuk dilakukannya uji terhadap data yang sudah di klasifikasi pada algortima. Splitting juga dilakukan kepada kedua variabel yaitu vatiabel x atau features dan variabel y atau target. Masing- masing variabel ketika di split akan menghasilkan 4 variabel berbeda. Diantaranya:

x_train: Variabel ini digunakan untuk data yang akan di train pada feature

x_test: Variabel ini digunakan untuk data yang akan di test pada feature

y_train: Variabel ini digunakan untuk data yang akan di train pada target

y_test: Variabel ini digunakan untuk data yang akan di test pada target

Split data pada tugas kali ini menggunakan test_size=0.2 yang artinya 20% dari data asli yang nantinya akan digunakan pada variabel test dan variabel train akan mendapati sisa dari data test atau 80% dari data asli.

3. Bagaimana konsep dan cara kerja dari KNN dan Naive Bayes?

- KNN
KNN (K-Nearest Neighbors) bekerja dengan cara mencari k tetangga terdekat dari data yang ingin diprediksi (**training**), dan menentukan kelas data tersebut berdasarkan mayoritas kelas dari tetangga-tetangga tersebut (**Prediksi/uji**). Penentuan kelas ini lah yang nantinya akan menjadi hasil klasifikasi. Pada algoritma ini penentuan kelas ditentukan dari banyaknya $n_neighbors=n$, nilai n tergantung keperluan diisi value berapa.
- Naïve Bayes
Algoritma ini menggunakan teorema Bayes untuk menghitung probabilitas suatu data baru termasuk dalam kelas tertentu. Cara kerjanya pun dibagi menjadi 2 mirip seperti KNN yaitu tahap training dan Uji. Algoritma dilatih dengan data yang telah diberi label kelas. Data ini digunakan untuk menghitung probabilitas kemunculan setiap fitur pada setiap kelas. Saat ada data baru, algoritma menghitung probabilitas data tersebut termasuk dalam setiap kelas. Kelas dengan probabilitas tertinggi dipilih sebagai kelas data baru tersebut.

4. Pada hasil pengujian manakah yang paling baik antara KNN dengan Naive Bayes pada dataset ini?

Untuk menentukan manakah yang paling baik antara KNN dengan Naïve Bayes bisa dilihat dari hasil keakurasian serta presisi pada hasil klasifikasi menggunakan library sklearn.

Table 1 Perbandingan akurasi dan presisi pada algoritma

	KNN	Naïve Bayes
Akurasi	0.5	0.625
Presisi	0.25	0.5

Dari hasil perbandingan keakurasian dan presisi bisa disimpulkan bahwa algoritma Naïve Bayes memiliki nilai yang lebih tinggi dibandingkan dengan KNN. Pada keakurasian KNN hanya memiliki nilai 0.5 sedangkan naïve bayes memiliki keakurasian dengan nilai 0.625 ini artinya pada keakurasian algoritma naïve bayes lebih baik dibandingkan KNN. Pada presisi Naïve Bayes juga memiliki angka yang lebih unggul dibandingkan KNN dengan angka 0.5 sedangkan pada KNN hanya 0.25

Dapat disimpulkan bahwa algoritma Naïve Bayes memiliki keunggulan pada Akurasi dan presisi maka Naïve Bayes lebih baik dibandingkan KNN.

5. Apa perbedaan dari akurasi dan presisi?

A. Akurasi

Merupakan proporsi total prediksi yang benar, dihitung dari semua prediksi yang dibuat. Dihitung dengan rumus: $\text{Akurasi} = (\text{Jumlah Prediksi Benar} / \text{Jumlah Prediksi Keseluruhan}) \times 100\%$. Memberikan gambaran keseluruhan tentang seberapa baik algoritma dalam memprediksi dengan benar.

Dengan menggunakan library sklearn terdapat function `accuracy_score(data_aktual, data_prediksi)` yang dapat mempermudah penghitungan

B. Presisi

Merupakan proporsi prediksi positif yang benar, dihitung dari semua prediksi yang diklasifikasikan sebagai positif.

Dihitung dengan rumus: $\text{Presisi} = (\text{Jumlah Positif Benar} / \text{Jumlah Prediksi Positif}) \times 100\%$.
Mengukur seberapa tepat algoritma dalam mengidentifikasi data positif.

Dengan menggunakan library sklearn terdapat function *precision_score(data_aktual, data_prediksi)* yang dapat mempermudah penghitungan