Praktikum 09

```
df = pd.read_csv("/content/drive/MyDrive/praktikum_ml/praktikum09/data/data.csv")
```

- pd.read_csv() → Fungsi pandas untuk membaca file CSV.
- Path file → "/content/drive/..." adalah lokasi file CSV di Google Drive (digunakan di Google Colab).
- df = ... → Menyimpan hasil pembacaan CSV ke variabel df dalam bentuk DataFrame.

```
df = df.drop(columns=["id", "Unnamed: 32"], errors="ignore")
```

- df.drop(...) → Menghapus kolom dari DataFrame.
- columns=["id", "Unnamed: 32"] → Daftar kolom yang ingin dihapus.
- errors="ignore" → Jika kolom tidak ditemukan, abaikan saja (tidak terjadi error).

```
df.head()
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_r
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1

5 rows × 31 columns

- df.head() → Menampilkan 5 baris pertama dari DataFrame.
- Fungsinya → Untuk melihat contoh data, memastikan data terbaca dengan benar, dan mengecek struktur awal DataFrame.

```
df.tail()
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity
564	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0
565	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0
566	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0
567	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0
568	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0

5 rows × 31 columns

- df.tail() → Menampilkan 5 baris terakhir dari DataFrame.
- Fungsinya → Untuk melihat bagian akhir data dan memastikan tidak ada data aneh atau baris kosong di bagian bawah dataset.

```
1 df.shape
```

```
(569, 31)
```

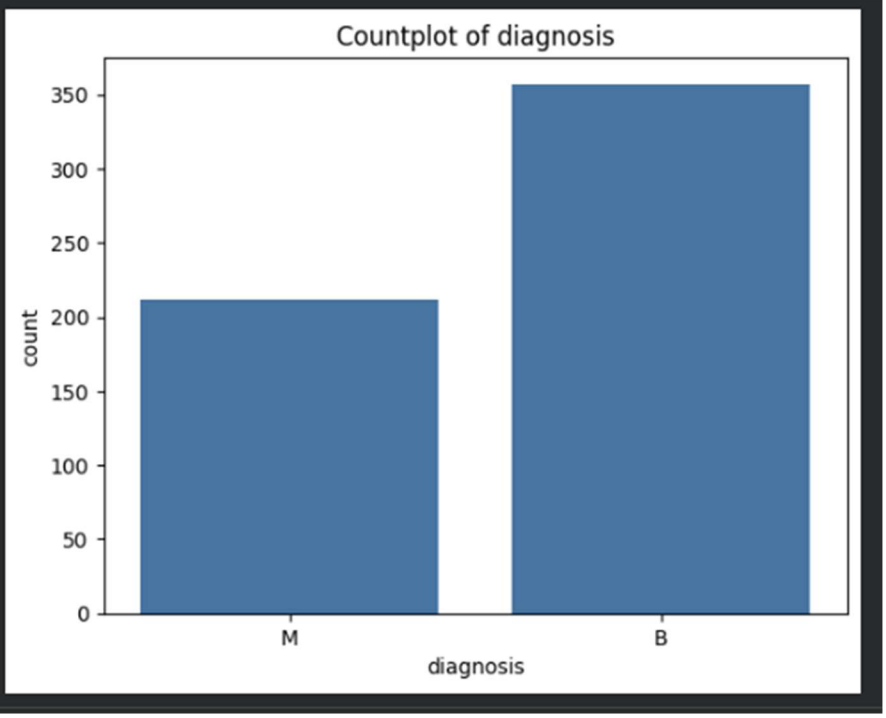
- df.shape → Mengambil informasi bentuk (dimensi) DataFrame.
- Hasilnya → Menghasilkan tuple (jumlah_baris, jumlah_kolom).
- Fungsinya → Untuk mengetahui ukuran dataset, berapa banyak baris dan kolom yang dimiliki.

```
1 df.isnull().sum()
```

radius_mean	0
texture_mean	0
perimeter_mean	0
area_mean	0
smoothness_mean	0

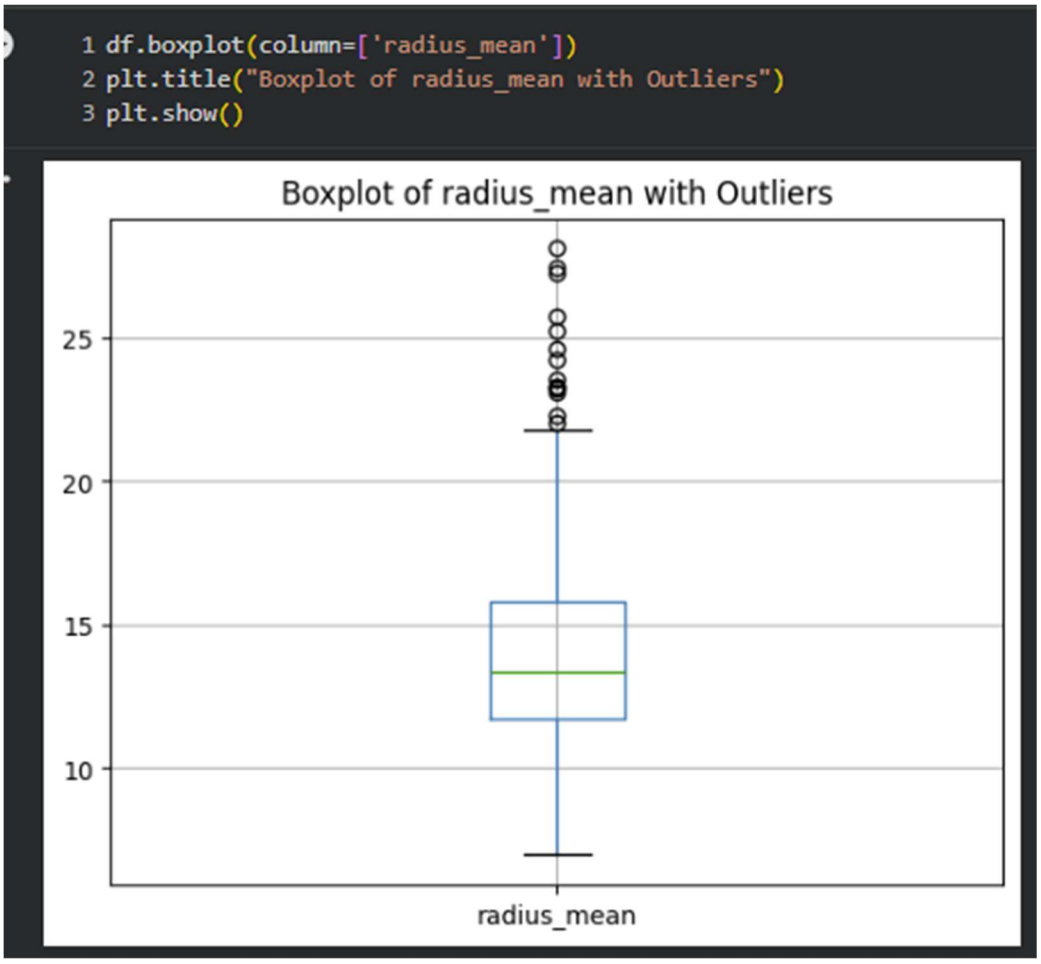
- df.isnull() → Mengecek apakah tiap sel berisi nilai kosong (NaN).
- .sum() → Menjumlahkan nilai kosong pada setiap kolom.
- Hasilnya → Menampilkan jumlah missing values untuk tiap kolom.
- Fungsinya → Untuk mengetahui kolom mana yang memiliki data hilang dan berapa banyak.

```
1 sns.countplot(x='diagnosis', data=df)
2 plt.title("Countplot of diagnosis")
3 plt.show()
```



- sns.countplot(x='diagnosis', data=df) -> Membuat grafik batang untuk menghitung jumlah setiap kategori pada kolom diagnosis. Menggunakan library Seaborn.
- plt.title("Countplot of diagnosis") -> Menambahkan judul pada grafik.
- plt.show() -> Menampilkan grafik ke layar.

```
df.boxplot(column=['radius_mean'])
plt.title("Boxplot of radius_mean with Outliers")
plt.show()
```



- df.boxplot(column=['radius_mean']) -> Membuat boxplot untuk kolom radius_mean. Digunakan untuk melihat persebaran data dan mendeteksi outlier.
- plt.title("Boxplot of radius_mean with Outliers") -> Memberi judul pada grafik boxplot.
- plt.show() -> Menampilkan grafik ke layar.

```
df['radius_mean'].fillna(df['radius_mean'].median(), inplace=True)
df['texture_mean'].fillna(df['texture_mean'].median(), inplace=True)
```

- df['radius_mean'].fillna(df['radius_mean'].median(), inplace=True) -> Mencari nilai median dari kolom radius_mean. Mengisi nilai yang kosong (NaN) di kolom tersebut dengan nilai median.
- inplace=True → perubahan dilakukan langsung pada DataFrame tanpa membuat variabel baru.
- df['texture_mean'].fillna(df['texture_mean'].median(), inplace=True) -> Melakukan hal yang sama seperti baris pertama, tetapi untuk kolom texture_mean.

```
1 df.describe()
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800

df.describe() -> Menghasilkan ringkasan statistik deskriptif untuk kolom numerik dalam DataFrame.

```
1 df['perimeter_mean'].value_counts()
```

perimeter_mean	count
82.61	3
134.70	3
87.76	3
113.40	2
120.20	2
...	...
82.53	1
100.40	1
81.15	1
60.73	1
87.02	1

522 rows × 1 columns

- df['perimeter_mean'] -> Mengakses kolom perimeter_mean pada DataFrame.
- .value_counts() -> Menghitung berapa banyak setiap nilai unik muncul dalam kolom tersebut. Hasilnya berupa daftar nilai unik beserta jumlah kemunculannya.

```
1 df['area_mean'].value_counts()
```

area_mean	count
512.2	3
321.6	2
582.7	2
1138.0	2
477.3	2
...	...
1148.0	1
642.7	1
461.0	1
951.6	1
477.4	1

539 rows × 1 columns

dtype: int64

- df['area_mean'] -> Mengakses kolom area_mean dari DataFrame.

```
1 df['smoothness_mean'].value_counts()
```

smoothness_mean	count
0.10070	5
0.11500	4
0.10540	4
0.10750	4
0.10630	3
...	...
0.08523	1
0.07371	1
0.12000	1
0.07941	1
0.08871	1

474 rows × 1 columns

dtype: int64

- df['smoothness_mean'] -> Mengakses kolom smoothness_mean di DataFrame.

```python
data = df

fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(20, 15))
axes = axes.flatten()

# Plot 1 – jumlah diagnosis
sns.countplot(x='diagnosis', data=data, ax=axes[0])
axes[0].set_title('Diagnosis Count (M/B)')

# Plot 2 – diagnosis dengan hue: radius_mean kategori
data['radius_cat'] = pd.qcut(data['radius_mean'], 3, labels=['Small', 'Medium', 'Large'])

sns.countplot(x='radius_cat', data=data, ax=axes[1])
axes[1].set_title('Radius Mean Category')

# Plot 3 – diagnosis berdasarkan radius_cat
sns.countplot(x='radius_cat', hue='diagnosis', data=data, ax=axes[2])
axes[2].set_title('Diagnosis by Radius Category')

# Plot 4 – diagnosis berdasarkan texture_mean (dibuat kategori)
data['texture_cat'] = pd.qcut(data['texture_mean'], 3, labels=['Low', 'Medium', 'High'])

sns.countplot(x='texture_cat', hue='diagnosis', data=data, ax=axes[3])
axes[3].set_title('Diagnosis by Texture Category')

# Plot 5 – diagnosis vs compactness_mean kategori
data['compact_cat'] = pd.qcut(data['compactness_mean'], 3, labels=['Low', 'Medium', 'High'])

sns.countplot(x='compact_cat', hue='diagnosis', data=data, ax=axes[4])
axes[4].set_title('Diagnosis by Compactness Category')

# Plot 6 – diagnosis vs symmetry_mean kategori
data['symmetry_cat'] = pd.qcut(data['symmetry_mean'], 3, labels=['Low', 'Medium', 'High'])

sns.countplot(x='symmetry_cat', hue='diagnosis', data=data, ax=axes[5])
axes[5].set_title('Diagnosis by Symmetry Category')

plt.tight_layout()
plt.show()
```

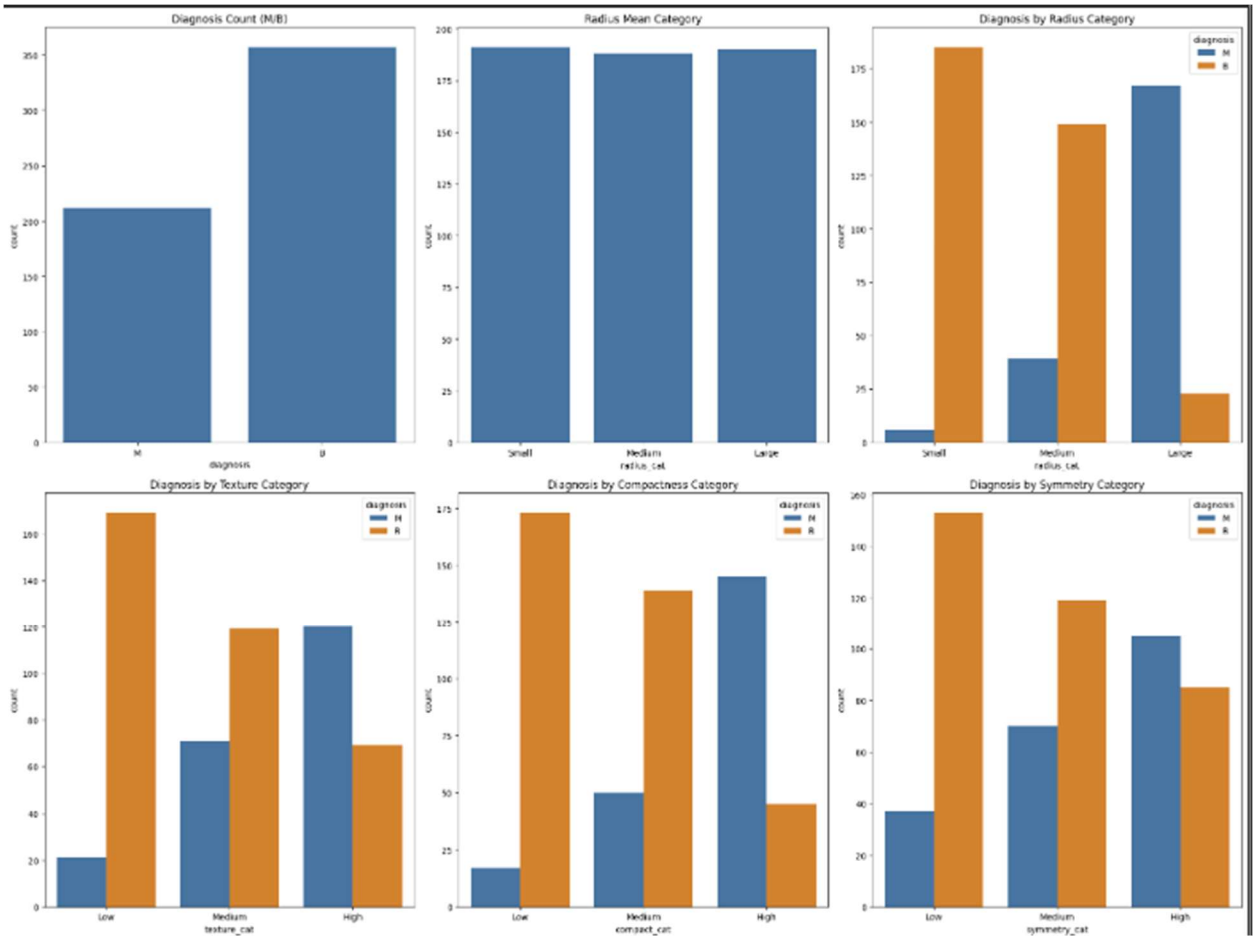
Persiapan Plot

data = df -> Menyalin DataFrame ke variabel data untuk memudahkan penggunaan.

fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(20, 15)) -. Membuat kanvas grafik dengan 2 baris × 3 kolom (6 subplot).

Ukuran gambar besar (20×15).

axes = axes.flatten() -> Mengubah matrix axes 2×3 menjadi array 1 dimensi agar mudah diakses: axes[0], axes[1], dst.



Plot 1 — Jumlah Diagnosis

- sns.countplot(x='diagnosis', data=data, ax=axes[0]) -> Menghitung dan menampilkan jumlah diagnosis M dan B.
- axes[0].set_title('Diagnosis Count (M/B)') -> Memberi judul pada plot.

Plot 2 — Kategori radius_mean

- data['radius_cat'] = pd.qcut(data['radius_mean'], 3, labels=['Small', 'Medium', 'Large'])
    - Mengubah nilai radius_mean menjadi 3 kategori berdasarkan kuantil (pembagian 3 bagian sama jumlah).
    - Label kategori: Small, Medium, Large.
- sns.countplot(x='radius_cat', data=data, ax=axes[1]) -> Menampilkan jumlah data pada tiap kategori radius.
- axes[1].set_title('Radius Mean Category') -> Judul plot.

Plot 3 — Diagnosis berdasarkan kategori radius

- sns.countplot(x='radius_cat', hue='diagnosis', data=data, ax=axes[2])
    - Menampilkan hubungan kategori radius_mean vs diagnosis.
    - Hue → membedakan warna untuk M/B.
- axes[2].set_title('Diagnosis by Radius Category')

Plot 4 — Diagnosis berdasarkan kategori texture_mean

- data['texture_cat'] = pd.qcut(data['texture_mean'], 3, labels=['Low', 'Medium', 'High']) -> Mengubah texture_mean menjadi 3 kategori kuantil.
- sns.countplot(x='texture_cat', hue='diagnosis', data=data, ax=axes[3]) -> Menampilkan hubungan kategori texture_mean vs diagnosis.
- axes[3].set_title('Diagnosis by Texture Category')

Plot 5 — Diagnosis berdasarkan compactness_mean

- data['compact_cat'] = pd.qcut(data['compactness_mean'], 3, labels=['Low', 'Medium', 'High']) -> Mengubah compactness_mean menjadi kategori kuantil.
- sns.countplot(x='compact_cat', hue='diagnosis', data=data, ax=axes[4])
- axes[4].set_title('Diagnosis by Compactness Category')

Plot 6 — Diagnosis berdasarkan symmetry_mean

- data['symmetry_cat'] = pd.qcut(data['symmetry_mean'], 3, labels=['Low', 'Medium', 'High']) -> Mengubah symmetry_mean menjadi 3 kategori kuantil.
- sns.countplot(x='symmetry_cat', hue='diagnosis', data=data, ax=axes[5])
- axes[5].set_title('Diagnosis by Symmetry Category')

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y, random_state=42)
```

Kode tersebut membagi data menjadi training 80% dan testing 20%, dengan pembagian kelas yang seimbang (menggunakan `stratify`) dan hasil pembagian yang konsisten karena memakai `random_state=42`.

```
print(X.shape, X_train.shape, X_test.shape)
```

```
(418, 7) (334, 7) (84, 7)
```

Pembagian data dengan proporsi 80% train dan 20% test.

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

- fit_transform pada X_train → menghitung mean & standar deviasi, lalu men-scale data training.
- transform pada X_test → men-scale data testing menggunakan parameter dari training (bukan dihitung ulang).

```python
from sklearn.naive_bayes import GaussianNB
nb_model = GaussianNB()
nb_model.fit(X_train_scaled, Y_train)
```

```
GaussianNB()
```

Kode tersebut membuat dan melatih model Gaussian Naive Bayes menggunakan data training yang sudah dinormalisasi.

Model nb_model kini siap digunakan untuk melakukan prediksi.

```python
# Accuracy
train_pred_nb = nb_model.predict(X_train_scaled)
test_pred_nb = nb_model.predict(X_test_scaled)

print("Training Accuracy (NB): ", accuracy_score(Y_train, train_pred_nb))
print("Testing Accuracy (NB): ", accuracy_score(Y_test, test_pred_nb))
```

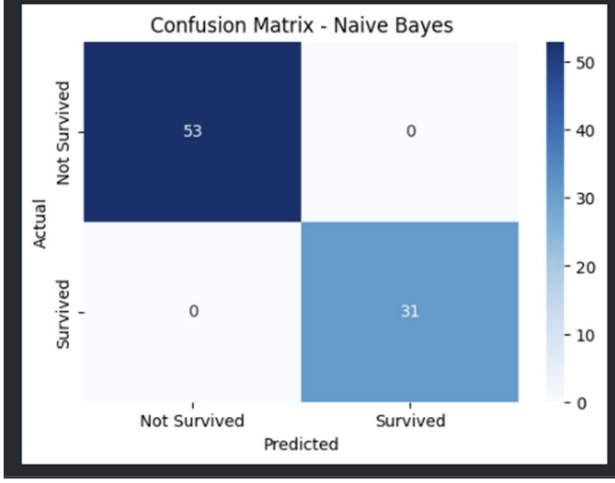
```
Training Accuracy (NB):  1.0
Testing Accuracy (NB):  1.0
```

Model Gaussian Naive Bayes berhasil mencapai akurasi 100% pada data training maupun testing, yang berarti model mampu mengklasifikasikan seluruh data dengan benar tanpa kesalahan.

```python
# Visualisasi Confusion Matrix (Naive Bayes)
plt.figure(figsize=(6,4))
cm_nb = confusion_matrix(Y_test, test_pred_nb)

sns.heatmap(cm_nb, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Not Survived', 'Survived'],
            yticklabels=['Not Survived', 'Survived'])

plt.title("Confusion Matrix - Naive Bayes")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



Kode tersebut menampilkan confusion matrix untuk model Naive Bayes dalam bentuk heatmap, sehingga terlihat jumlah prediksi benar dan salah pada data testing.

Hasilnya menunjukkan performa model secara visual antara actual dan predicted untuk kelas Not Survived dan Survived.

```
print("\nClassification Report (NB):")
print(classification_report(Y_test, test_pred_nb))
```

```
Classification Report (NB):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        53
           1       1.00      1.00      1.00        31

    accuracy                           1.00        84
   macro avg       1.00      1.00      1.00        84
weighted avg       1.00      1.00      1.00        84
```

Kode ini menampilkan classification report untuk model Naive Bayes, yang berisi metrik evaluasi:

- Precision → akurasi prediksi positif.
- Recall → kemampuan model menemukan semua data positif.
- F1-score → rata-rata harmonis antara precision dan recall.
- Support → jumlah sampel tiap kelas.

Dengan akurasi 100%, semua metrik akan menunjukkan nilai 1.0, artinya model sempurna pada data testing.

```
from sklearn.model_selection import cross_val_score
cv_nb = cross_val_score(nb_model, X, Y, cv=5, scoring='accuracy')
print("\nNaive Bayes Cross Validation Accuracy (5-Fold):")
print("Scores:", cv_nb)
print("Mean Accuracy:", cv_nb.mean())
print("Std Deviation:", cv_nb.std())
```

```
Naive Bayes Cross Validation Accuracy (5-Fold):
Scores: [1. 1. 1. 1. 1.]
Mean Accuracy: 1.0
Std Deviation: 0.0
```

Model Naive Bayes menunjukkan performa sangat stabil dan sempurna saat diuji dengan 5-fold cross-validation:

Semua skor akurasi fold = 1.0

Rata-rata akurasi = 1.0

Deviasi standar = 0.0, menunjukkan tidak ada variasi, model konsisten di semua subset data.