BAB II

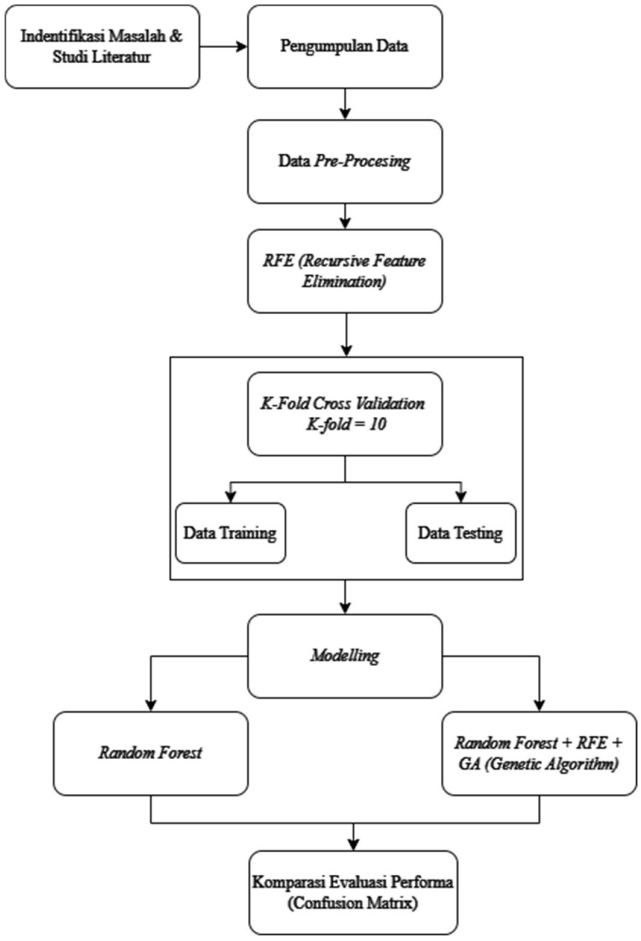
METODOLOGI PENELITIAN

2.1 Objek Penelitian

Objek pada penelitian ini adalah kasus stunting di Kota Samarinda. Data yang digunakan diperoleh dari Dinas Kesehatan Kota Samarinda, dimana pada tahun 2021, prevalensi stunting pada balita di kota tersebut tercatat sebesar 24,7% (Rufina et., al 2023). Data kasus stunting ini dikumpulkan dari 26 puskesmas yang tersebar di berbagai wilayah Kota Samarinda, semuanya beroperasi di bawah pengawasan Dinas Kesehatan setempat. Proses pengumpulan data dilakukan dengan mendokumentasikan kasus-kasus stunting yang teridentifikasi oleh petugas kesehatan di puskesmas-puskesmas tersebut. Penelitian ini dilakukan langsung di Dinas Kesehatan Kota Samarinda, yang berlokasi di Jl. Milono No.1, Bugis, Kecamatan Samarinda Kota, Kota Samarinda, Kalimantan Timur, 76112, untuk memastikan keakuratan dan integritas data yang dikumpulkan

2.2 Teknik Analisis Data

Penelitian ini bertujuan untuk merancang sebuah kerangka kerja metodologis yang rapi dan sistematis, memastikan bahwa setiap tahap penelitian dilakukan secara konsisten dan terstruktur. Pendekatan analitis yang dipilih telah dirancang untuk mengikuti sebuah prosedur yang terorganisir dengan jelas, yang akan dijelaskan lebih lanjut dalam alur penelitian berikut:



Gambar 2. 1 Alur Penelitian

2.2.1 Pengumpulan data

Langkah pertama dalam penelitian ini adalah memahami data yang berkaitan dengan kasus Stunting, yang dimulai dengan pengumpulan data dan diikuti oleh persiapan data. Data yang terkumpul terdiri dari 27 atribut. Tabel 2.1 menunjukkan data yang telah diperoleh dari Dinas Kesehatan Kota Samarinda.

Tabel 2. 1 Atribut Data Stunting Dinas Kesehatan Kota Samarinda

No	Atribut	Tipe Data	Keterangan
1	Nama	*String*	Nama
2	JK	*String*	Jenis Kelamin
3	BB Lahir	*Integer*	Berat Bayi Lahir
4	TB Lahir	*Integer*	Tinggi Badan Bayi Lahir
5	Provinsi	*String*	Provinsi
6	Kab/Kota	*String*	Kabupaten atau Kota
7	Kec	*String*	Kecamatan
8	Puskesmas	*String*	Lokasi Puskesmas
9	Posyandu	*String*	Lokasi Posyandu
10	RT	*Integer*	Rukun Tetangga
11	RW	*Integer*	Rukun Warga
12	Usia Saat Ukur	*Integer*	Usia Bayi Saat Ukur
13	Tanggal Pengukuran	*Date*	Tanggal Pengukuran
14	Berat	*Integer*	Berat Badan
15	Tinggi	*Integer*	Tinggi Badan
16	LiLA	*Integer*	Lingkar Lengan Atas
17	BB/U	*String*	Berat Badan Menurut Umur
18	ZS BB/U	*Integer*	Z Score Berat Badan menurut Umur
19	TB/U	*String*	Tinggi Badan menurut Umur
20	ZS TB/U	*Integer*	Z Score Tinggi Badan menurut Umur
21	BB/TB	*String*	Berat Badan menurut Tinggi Badan
22	ZS BB/TB	*Integer*	Z Score Berat Badan menurut Tinggi Badan
23	Naik Berat Badan	*String*	Naik Berat Badan Bayi
24	PMT Diterima(kg)	*Integer*	Pemberian Makanan Tambahan Bayi
25	Jml Vit A	*Integer*	Jumlah Vitamin A
26	KPSP	*String*	Kuesioner Pra Skrining Perkembangan
27	KIA	*String*	Kartu Indentitas Anak

Tabel 2.1 menampilkan 27 atribut data stunting yang diperoleh dari Dinas Kesehatan Kota Samarinda. Atribut-atribut ini mencakup informasi demografis, pengukuran fisik, status gizi, dan pemberian makanan tambahan. Data tersebut akan digunakan sebagai dasar dalam proses seleksi fitur dan analisis stunting dalam penelitian ini.

2.2.2 Data Pre-Procesing

Dalam tahap ini, proses pre-processing data akan meliputi tiga tahapan kunci yang penting untuk mempersiapkan data agar siap dianalisis. Tahapan pertama adalah data selection, di mana data yang relevan akan dipilih dari keseluruhan kumpulan data berdasarkan kriteria yang telah ditentukan. Tahap kedua, data cleaning, melibatkan mengidentifikasi dan memperbaiki atau menghapus data yang rusak atau tidak lengkap. Tahap terakhir, pada tahap data transformation, data akan diubah atau dikonversi ke

format yang lebih sesuai untuk analisis, termasuk normalisasi atau standarisasi nilai. Data yang digunakan dalam penelitian ini diperoleh dari Dinas Kesehatan Kota Samarinda, dan telah melalui proses persiapan yang telah ditetapkan sebagai berikut:

a) Data Selection

Dalam tahap ini, proses pengambilan data melibatkan pemilihan fitur atau atribut yang relevan untuk penelitian, sementara fitur yang dianggap tidak relevan akan dieliminasi. Sebagai contoh, data awal yang diperoleh dari Dinas Kesehatan Kota Samarinda, yang tertera dalam Tabel 2.2, mencakup 27 atribut. Setelah proses seleksi fitur, 14 atribut dianggap tidak relevan dan dihapus, sehingga tersisa 12 atribut yang akan digunakan sebagai fitur dan 1 atribut lainnya yang akan dijadikan kelas atau target dalam klasifikasi stunting pada anak. Data yang telah diseleksi tersebut ditampilkan pada Tabel 2.2.

Tabel 2. 2 Data Selection

No	Atribut	Tipe Data	Keterangan
1	Nama	*String*	Nama Balita
2	JK	*String*	Jenis Kelamin
3	Berat	*Integer*	Berat Badan
4	Tinggi	*Integer*	Tinggi Badan
5	LiLA	*Integer*	Lingkar Lengan Atas
6	BB/U	*String*	Berat Badan Menurut Umur
7	ZS BB/U	*Integer*	Z Score Berat Badan menurut Umur
8	TB/U	*String*	Tinggi Badan menurut Umur
9	ZS TB/U	*Integer*	Z Score Tinggi Badan menurut Umur
10	BB/TB	*String*	Berat Badan menurut Tinggi Badan
11	ZS BB/TB	*Integer*	Z Score Berat Badan menurut Tinggi Badan
12	Naik Berat B	*String*	Naik Berat Badan
13	Tanggal Pengukuran	*Date*	Tanggal Pengukuran

Data yang telah diseleksi pada Tabel 2.2 mencakup 13 atribut, di mana 12 atribut akan digunakan sebagai fitur dan 1 sebagai target dalam klasifikasi stunting pada anak. Atribut-atribut ini mencakup informasi relevan seperti berat badan, tinggi badan, lingkar lengan atas, dan berbagai z-score yang mengukur status gizi anak. Proses seleksi fitur ini bertujuan menghilangkan atribut yang kurang signifikan untuk meningkatkan efisiensi dan akurasi model klasifikasi, sehingga dapat lebih efektif dalam mengidentifikasi faktor-faktor penting yang berkontribusi terhadap stunting dan menghasilkan prediksi yang lebih akurat.

b) Data Cleaning

Dalam tahap ini, data yang akan digunakan akan diolah melalui proses pembersihan, di mana data yang memiliki nilai #N/A (no value is available) atau data yang tidak memiliki nilai, serta data yang terduplikasi, akan dihapus. Untuk melaksanakan proses ini, kami akan menggunakan bahasa pemrograman Python bersama dengan library Pandas. Khususnya, fungsi dropna() akan digunakan untuk menggantikan nilai yang hilang dengan nilai rata-rata dari data tersebut.

```
# Menghapus baris yang memiliki setidaknya satu nilai yang hilang
data_tanpa_isi = data.dropna()
```

Gambar 2. 2 Kode *Cleaning* Data Kosong

c) Data Transformation

Dalam tahap ini, nilai-nilai dari atribut kategorikal akan dikonversi menjadi bentuk numerik. Konversi ini diperlukan karena *library sklearn* yang digunakan hanya menerima atribut dalam format numerik. Atribut yang akan ditransformasi mencakup JK, Berat, Tinggi, LiLA, BB/U ZS BB/U, TB/U, ZS TB/U, BB/TB, ZS BB/TB, Kelas. Proses transformasi ini akan dilakukan menggunakan bahasa pemrograman Python dan library pandas, dengan menerapkan fungsi map untuk pemetaan nilai-nilai tersebut.

```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder, OrdinalEncoder

# Membaca dataset
stunting = pd.read_csv('dataset_stunting_samarinda_terbaru.csv')

# Menampilkan kolom yang ada dalam DataFrame
print("Kolom dalam DataFrame:", stunting.columns)

# Membuat instance untuk encoder
ordinal = OrdinalEncoder()
labelencoder = LabelEncoder()

# Daftar kolom yang dibutuhkan
columns_needed = ['JK', 'Berat', 'Tinggi', 'LiLA', 'BB/U', 'ZS BB/U', 'ZS TB/U', 'Naik Berat Badan', 'BB/TB', 'ZS BB/TB', 'Kelas']

# Menyimpan subset data yang diperlukan
df_transform = stunting[columns_needed]

# Transformasi data kategorikal menjadi numerik
stunting['JK'] = labelencoder.fit_transform(stunting['JK'])
stunting['BB/U'] = labelencoder.fit_transform(stunting['BB/U'])
stunting['BB/TB'] = labelencoder.fit_transform(stunting['BB/TB'])
stunting['Naik Berat Badan'] = labelencoder.fit_transform(stunting['Naik Berat Badan'])
```

Gambar 2. 3 Kode Data *Transformation*

2.2.3 K-Fold Cross Validation

Sebelum memulai pembuatan model, dataset yang digunakan perlu dibagi menjadi dua bagian pokok, yakni data pelatihan dan data pengujian. Data pelatihan digunakan sebagai dasar untuk mengembangkan model, sedangkan data pengujian bertugas untuk menilai efektivitas model yang telah dibangun. Dalam penelitian ini, teknik *K-Fold Cross Validation* akan digunakan, yang direalisasikan menggunakan *library sklearn.model_selection* dengan fungsi *cross_val_score* pada *Python*. Teknik ini membagi data menjadi 10 bagian atau kelompok, berdasarkan parameter Cv yang ditetapkan sebesar 10, yang berarti dataset dibagi menjadi 10 segmen. Setiap segmen akan secara bergantian digunakan sebagai data latih dan data uji. Dengan nilai k yang ditetapkan sebesar 10, eksperimen akan dijalankan sepuluh kali dan nilai rata-rata dari semua hasil pengujian akan digunakan untuk menghasilkan estimasi yang lebih presisi dan akurat.

```python
# Membagi data menjadi training dan testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Membuat 10-fold cross-validator
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
```

Gambar 2. 4 Kode *K-Fold Cross Validation*

Berikut disajikan Tabel 2.3 yang mencakup setiap parameter yang terlibat dalam kode beserta deskripsi fungsi masing-masing parameter:

Tabel 2. 3 Penjelasan Parameter Pembagian Dataset dan Perulangan

Parameter	Keterangan
X	Data fitur yang digunakan untuk membuat prediksi.
y	Kolom target dari dataset, digunakan sebagai label dalam pemodelan.
n_splits	Jumlah pembagian data dalam proses cross-validation, menunjukkan berapa kali data dibagi. Misalnya, n_splits=10 menunjukkan bahwa data akan dibagi menjadi 10 bagian.
shuffle	Menentukan apakah sampel dalam dataset harus diacak sebelum dibagi menjadi batch untuk cross-validation. Pengacakan membantu dalam menghindari bias sampling dan biasanya diatur menjadi True untuk mendistribusikan data secara merata.
random_state	Parameter yang digunakan untuk mengontrol keacakan saat mengacak atau membagi data, yang membantu dalam mencapai hasil yang konsisten dan dapat direproduksi. Misalnya, mengatur 'random_state=42' berarti bahwa setiap kali kode dijalankan, pemisahan data akan sama.

2.2.4 Pembuatan Model

Berikut adalah deskripsi dan rincian dari model klasifikasi Random Forest yang diimplementasikan dalam penelitian ini :

a) Dalam tahap ini, dataset dibagi menjadi data *training* dan data *testing*. Data pelatihan digunakan untuk mengembangkan model, sementara data pengujian untuk menilai kinerjanya. Menggunakan teknik K-Fold Cross Validation dengan 10 lipatan *melalui library sklearn.model_selection* dan fungsi *cross_val_score*, model *Random Forest* diterapkan untuk klasifikasi stunting di Kota Samarinda. Berikut disajikan penjelasan tentang cara kerja metode ini:

1. Tahap pertama, penelitian ini melibatkan proses ekstraksi sampel dari dataset yang berkaitan dengan stunting di Kota Samarinda. Untuk mengumpulkan data, akan digunakan entri dengan tanggal pengukuran stunting yang paling terkini, sementara entri dengan nama yang sama tetapi dengan tanggal pemeriksaan yang lebih lama akan dieliminasi. Hal ini dilakukan untuk memastikan data yang diperoleh mencerminkan hasil terakhir dari pemeriksaan stunting tahun 2023.

```
# Mengurutkan data berdasarkan kolom "Nama" dan "Tanggal Pengukuran" secara menurun
data = data.sort_values(by=['Nama', 'Tanggal Pengukuran'], ascending=[True, False])

# Menghapus duplikat berdasarkan kolom "Nama" dan mempertahankan yang pertama (yang memiliki tanggal pengukuran terbaru)
data = data.drop_duplicates(subset='Nama', keep='first')
```

Gambar 2. 5 Kode Data Pengukuran Terbaru

2. Tahap kedua, Inisialisasi Model *Random Forset*. Pada proses pembuatan pohon ini, teknik *Classification and Regression Tree (CART)* diterapkan, dengan menggunakan *gini impurity* sebagai kriteria utama untuk menentukan pembagian di setiap kode dalam pohon.

```
from sklearn.ensemble import RandomForestClassifier

rf_classifier = RandomForestClassifier(
    n_estimators=10 + i,  # Mengubah jumlah estimators
    criterion='gini',
    max_depth=1 + i % 3,  # Mengubah kedalaman maksimal pohon
    min_samples_split=100 + i,  # Mengubah jumlah minimal sampel untuk split
    min_samples_leaf=750 - i * 5,  # Mengubah jumlah minimal sampel per daun
    max_features=0.1 + (i % 5) * 0.02,  # Mengubah jumlah fitur yang dipertimbangkan untuk split
    class_weight='balanced',
    random_state=42 + i  # Mengubah random state
)
```

Gambar 2. 6 Kode *Random Forest*

3. Tahap ketiga proses di langkah b diulang terus menerus sampai jumlah pohon keputusan yang ditargetkan, yaitu k = 10, tercapai. Dan membagi dataset menjadi data *training* dan *testing.*

4. Tahap keempat dalam pengembangan model menggunakan algoritma *Random Forest* melibatkan aplikasi library *scikit-learn*. Pada tahap ini, teknik *k-fold* cross-validation digunakan untuk memisahkan dataset menjadi bagian latihan dan uji, serta untuk menilai kinerja model dengan cara yang objektif. Awalnya, model *Random Forest* dibuat menggunakan *Random Forest Classifier* dari *scikit-learn*. Kemudian, proses iterasi dilakukan melalui tiap fold yang dibentuk oleh *k-fold cross-validation.* Dalam setiap iterasi, dataset dibagi antara data latihan dan uji, dengan model *Random Forest* dilatih menggunakan data latihan. Setelah proses pelatihan, model ini diaplikasikan untuk membuat prediksi pada data uji. Skor evaluasi seperti akurasi, presisi, recall, dan F1-score dihitung menggunakan metrik yang relevan dari library *sklearn.metrics* dan dicatat untuk tiap *fold*. Selain itu, *confusion matrix* untuk tiap *fold* juga dihitung menggunakan fungsi *confusion_matrix* dari *scikit-learn.* Prediksi dan skor evaluasi ini lalu disimpan untuk analisis lebih lanjut mengenai kinerja model terhadap dataset yang dipilih. setelah menyelesaikan proses *k-fold cross-validation* dan menguji model *Random Forest* pada setiap *fold*, skor evaluasi seperti akurasi, presisi, *recall*, dan *F1 Score* untuk tiap *fold* akan ditampilkan. Menggunakan perulangan dengan *`enumerate`* dan *`zip`*, skor evaluasi tiap fold diproses dan ditampilkan pada layar. Selanjutnya, nilai rata-rata dari skor evaluasi semua fold dihitung menggunakan fungsi *`np.mean()`* dari *NumPy* dan hasilnya dicetak. Selain itu, rata-rata dari matriks kebingungan *(confusion matrix)* untuk semua *fold* dihitung dan ditampilkan menggunakan fungsi yang sama. Matriks kebingungan rata-rata ini memberikan indikasi tentang efektivitas model dalam mengklasifikasikan tiap kelas dalam data uji.

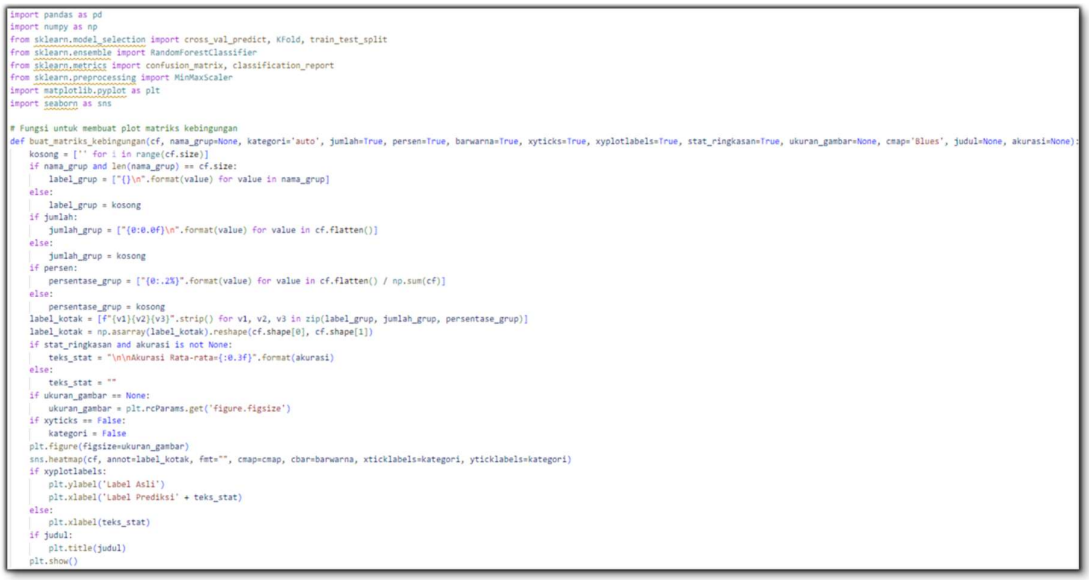
$$\mathcal{Y} = argmax\mathcal{X} \sum_{i=1}^{T} I(\mathrm{h}i(\mathcal{X}) = \mathcal{K}) \qquad (2.1)$$

$\mathcal{Y}$ = Kelas perdiksi akhir untuk data $\mathcal{X}$

$argmax\mathcal{X}$ = Operator yang mencari nilai $\mathcal{K}$ yang memaksimalkan jumlah suara (argument maksium). Dalam konteks ini, $\mathcal{K}$ adalah kelas.

$\sum_{i=1}^{T}$ = Menunjukan penjumlahan dari i =1 hingga T, dimana T adalah jumlah total pohon dalam random Forest.

$I(\mathrm{h}i(\mathcal{X}) = \mathcal{K})$ = Fungsi indicator yang bernilai 1 jika diprediksi pohon ke- i (h i) untuk data $\mathcal{X}$ adalah kelas $\mathcal{K}$, dan 0 jika tidak.



```
import pandas as pd
import numpy as np
from sklearn.model_selection import cross_val_predict, KFold, train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import seaborn as sns

# Fungsi untuk membuat plot matriks kebingungan
def buat_matriks_kebingungan(cf, nama_grup=None, kategori='auto', jumlah=True, persen=True, barwarna=True, xyticks=True, xyplotlabels=True, stat_ringkasan=True, ukuran_gambar=None, cmap='Blues', judul=None, akurasi=None):
    kosong = ['' for i in range(cf.size)]
    if nama_grup and len(nama_grup) == cf.size:
        label_grup = ["{}\n".format(value) for value in nama_grup]
    else:
        label_grup = kosong
    if jumlah:
        jumlah_grup = ["{0:0.0f}\n".format(value) for value in cf.flatten()]
    else:
        jumlah_grup = kosong
    if persen:
        persentase_grup = ["{0:.2%}".format(value) for value in cf.flatten() / np.sum(cf)]
    else:
        persentase_grup = kosong
    label_kotak = [f"{v1}{v2}{v3}".strip() for v1, v2, v3 in zip(label_grup, jumlah_grup, persentase_grup)]
    label_kotak = np.asarray(label_kotak).reshape(cf.shape[0], cf.shape[1])
    if stat_ringkasan and akurasi is not None:
        teks_stat = "\n\nAkurasi Rata-rata={:0.3f}".format(akurasi)
    else:
        teks_stat = ""
    if ukuran_gambar == None:
        ukuran_gambar = plt.rcParams.get('figure.figsize')
    if xyticks == False:
        kategori = False
    plt.figure(figsize=ukuran_gambar)
    sns.heatmap(cf, annot=label_kotak, fmt="", cmap=cmap, cbar=barwarna, xticklabels=kategori, yticklabels=kategori)
    if xyplotlabels:
        plt.ylabel('Label Asli')
        plt.xlabel('Label Prediksi' + teks_stat)
    else:
        plt.xlabel(teks_stat)
    if judul:
        plt.title(judul)
    plt.show()
```

```
# Membaca data yang telah diproses
data = pd.read_csv('dataset_stunting_samarinda_modif.csv')

# Mengganti nama kolom 'TB/U' menjadi 'Kelas'
data = data.rename(columns={'TB/U': 'Kelas'})

# Menghapus kolom non-numerik dan tidak relevan
data = data.drop(['Nama', 'Tanggal Pengukuran'], axis=1)

# Mengganti nilai bermasalah di kolom 'JK'
data['JK'] = data['JK'].str.strip().replace({'L\xa0': 'L', 'P\xa0': 'P'})

# Menangani nilai numerik bermasalah
data['Berat'] = data['Berat'].str.replace(',', '.').astype(float)

# Mengonversi kolom kategorikal menjadi numerik menggunakan one-hot encoding
kolom_kategorikal = ['JK', 'BB/U', 'BB/TB', 'Naik Berat Badan']
data = pd.get_dummies(data, columns=kolom_kategorikal, drop_first=True)

# Memastikan tidak ada kehilangan data setelah konversi
print(f"Bentuk dataset setelah konversi: {data.shape}")

# Memisahkan fitur dan variabel target
X = data.drop('Kelas', axis=1)
y = data['Kelas'].map({'Tidak Stunting': 0, 'Stunting': 1})

# Menormalisasi fitur
scaler = MinMaxScaler()
X_normalized = scaler.fit_transform(X)

# Membagi data menjadi training dan testing set
x_train, x_test, y_train, y_test = train_test_split(X_normalized, y, test_size=0.2, random_state=42)

# Merekam hasil K-Fold 10 kali
kfolds = 10
repeats = 10
akurasi_list = []

# Menggunakan KFold untuk cross-validation
kf = KFold(n_splits=kfolds, shuffle=True, random_state=1)
fold = 1
```

```
# Melakukan cross-validation dengan variasi
for i in range(repeats):
    print(f"Rekaman {i+1}")

    # Mendefinisikan classifier Random Forest dengan parameter yang sedikit berbeda untuk setiap iterasi
    rf_classifier = RandomForestClassifier(
        n_estimators=10 + i,  # Mengubah jumlah estimators
        criterion='gini',
        max_depth=1 + i % 3,  # Mengubah kedalaman maksimal pohon
        min_samples_split=100 + i,  # Mengubah jumlah minimal sampel untuk split
        min_samples_leaf=50 - i * 5,  # Mengubah jumlah minimal sampel per daun
        max_features=0.1 + (i % 5) * 0.02,  # Mengubah jumlah fitur yang dipertimbangkan untuk split
        class_weight='balanced',
        random_state=42 + i  # Mengubah random state
    )

    y_pred = cross_val_predict(rf_classifier, x_train, y_train, cv=kfolds)
    cf_matrix = confusion_matrix(y_train, y_pred)
    akurasi = np.trace(cf_matrix) / float(np.sum(cf_matrix))
    akurasi_list.append(akurasi)
    print(f"Akurasi: {akurasi:.3f}")
    class_report = classification_report(y_train, y_pred, target_names=['Tidak Stunting', 'Stunting'])
    print(class_report)

# Menggambar plot matriks kebingungan untuk iterasi terakhir dengan rata-rata akurasi
buat_matriks_kebingungan(cf_matrix, ukuran_gambar=(8, 6), barwarna=False, judul=f"Model Random Forest (K-Fold = 10)")
```

Gambar 2. 7 Kode Model *Algoritma Random Forest*

Berikut adalah Tabel 2.4 yang menyajikan daftar parameter yang digunakan dalam kode tersebut, beserta deskripsi fungsi dari masing-masing parameter.

Tabel 2. 4 Parameter Model Algoritma *Random Forest*

Parameter	Keterangan
n_estimators=10	Mengatur jumlah pohon dalam hutan menjadi 10.
criterion='gini'	Menggunakan kriteria 'gini' untuk mengukur kualitas split.
max_depth=1	Membatasi kedalaman maksimum pohon menjadi 1 untuk mencegah overfitting.
min_samples_split=100	Jumlah minimum sampel yang diperlukan untuk memisahkan node internal.
min_samples_leaf=50	Jumlah minimum sampel yang diperlukan untuk berada di daun node.
max_features=0.1	Proporsi maksimum fitur yang dipertimbangkan untuk split, diatur ke 10%.
class_weight='balanced'	Mengatur bobot kelas untuk menangani ketidakseimbangan kelas.
random_state=42	Mengatur seed untuk memastikan hasil yang konsisten.
nama_grup	Daftar nama untuk setiap grup di matriks kebingungan.
kategori	Kategori untuk sumbu x dan y.
jumlah	Menampilkan jumlah sampel di setiap kotak matriks.
persen	Menampilkan persentase di setiap kotak matriks.
barwarna	Menampilkan bar warna pada plot.
xyticks	Menampilkan tick marks pada sumbu x dan y.
xyplotlabels	Menampilkan label untuk sumbu x dan y.
stat_ringkasan	Menampilkan statistik ringkasan pada plot.
ukuran_gambar	Ukuran gambar untuk plot.
cmap	Skema warna untuk plot.
scaler	Objek MinMaxScaler untuk normalisasi fitur.
X_normalized	Hasil normalisasi fitur-fitur dalam dataset.
kfolds	Jumlah split dalam cross-validation (10-fold).
repeats	Jumlah pengulangan cross-validation (10 kali).
kf	Objek KFold untuk melakukan cross-validation.
fold	Nomor fold dalam cross-validation (tidak digunakan lebih lanjut dalam kode).
y_pred	Hasil prediksi dari cross-validation.
cf_matrix	Matriks kebingungan yang dihasilkan dari hasil prediksi dan label asli.
akurasi	Akurasi model yang dihitung dari matriks kebingungan.
class_report	Laporan klasifikasi yang berisi metrik seperti precision, recall, dan f1-score.

b) Dalam tahap seleksi fitur menggunakan metode *Recursive Feature Elimination (RFE)* dilakukan dengan cara mengevaluasi dan menghilangkan atribut yang kurang signifikan secara bertahap untuk meningkatkan performa model klasifikasi. Metode ini memulai dengan set fitur lengkap dan kemudian secara sistematis menghilangkan fitur-fitur yang memiliki bobot atau pengaruh paling rendah terhadap variabel target berdasarkan kriteria tertentu, yang seringkali ditentukan oleh model yang digunakan, seperti koefisien dalam regresi atau pentingnya fitur dalam pohon keputusan. Secara spesifik, *RFE* bekerja dengan cara sebagai berikut:

1. Semua fitur yang relevan dipilih dari dataset stunting untuk dilakukan analisis lebih lanjut.
2. Dalam menggunakan metode *Recursive Feature Elimination (RFE)* yang diimplementasikan melalui library *scikit-learn*, metode ini memilih fitur dengan mengeliminasi fitur yang paling lemah satu per satu berdasarkan bobot yang ditentukan oleh estimator yang digunakan. Di bawah ini adalah konsep umum dari *RFE*:

```
from sklearn.feature_selection import RFE


# Melakukan seleksi fitur menggunakan RFE
selector = RFE(rf_base, n_features_to_select=2, step=1)
selector = selector.fit(X_normalized, y)
X_selected = selector.transform(X_normalized)
```

Gambar 2. 8 Kode Seleksi Fitur *RFE*

Berikut adalah Tabel 2.5 yang menyajikan daftar parameter yang digunakan dalam kode tersebut, beserta deskripsi fungsi dari masing-masing parameter:

Tabel 2. 5 Parameter Seleksi Fitur *RFE*

Parameter	Keterangan
rf_base	Model dasar yang digunakan oleh *RFE* untuk menilai pentingnya fitur (*RandomForestClassifier*).
n_features_to_select=2	Jumlah fitur yang diinginkan untuk dipilih oleh *RFE* (2 fitur).
step=1	Jumlah fitur yang dihilangkan pada setiap iterasi RFE (1 fitur per iterasi).

c) Dalam tahap optimasi menggunakan Algoritma *Genetika (GA)*, dilakukan dengan mengevaluasi dan memodifikasi atribut secara bertahap untuk meningkatkan performa model klasifikasi. Metode ini memulai dengan set fitur lengkap dan kemudian secara sistematis mengidentifikasi dan memodifikasi konfigurasi fitur yang memiliki pengaruh paling signifikan terhadap variabel target. Proses ini didasarkan pada kriteria tertentu yang seringkali ditentukan oleh hasil evaluasi *fitness* dalam *GA*. Secara spesifik, Algoritma *Genetika* bekerja dengan cara sebagai berikut:

1. Untuk mengimplementasikan optimasi menggunakan *Algoritma Genetik (GA)* sebagai lanjutan dari kode seleksi fitur *RFE* yang telah kita bahas, kita akan memerlukan library tambahan yang dapat menangani optimasi genetik. Salah satu library yang populer untuk ini adalah *'DEAP' (Distributed Evolutionary Algorithms in Python)*. Dalam contoh ini, kita akan menggunakan *DEAP* untuk mengoptimalkan fitur-fitur yang dipilih dari model *Random Forest Classifier* untuk mencapai akurasi yang lebih tinggi.
2. Berikut adalah kode yang menunjukkan bagaimana mengintegrasikan optimasi *GA* dengan model *Random Forest Classifier* yang sudah ada:

```
try:
    from deap import base, creator, tools, algorithms
except ImportError:
    !pip install deap
    from deap import base, creator, tools, algorithms

# Konfigurasi GA
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)

toolbox = base.Toolbox()
toolbox.register("attr_bool", np.random.randint, 0, 2)
toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_bool, n=len(X.columns))
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

# Fungsi evaluasi
def evalRF(individual):
    selected_features = [i for i, bit in enumerate(individual) if bit == 1]
    if len(selected_features) == 0:
        return 0,  # Hindari subset fitur kosong
    X_subset = X_normalized[:, selected_features]
    selector = RFE(rf_base, n_features_to_select=min(2, len(selected_features)), step=1)
    X_selected = selector.fit_transform(X_subset, y)
    y_pred = cross_val_predict(rf_base, X_selected, y, cv=5)  # Mengurangi jumlah CV fold
    return (np.mean(cross_val_predict(rf_base, X_selected, y, cv=5) == y),)  # Mengembalikan akurasi

toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)
toolbox.register("select", tools.selTournament, tournsize=3)
toolbox.register("evaluate", evalRF)

# Menjalankan GA dengan pengaturan yang lebih cepat
population = toolbox.population(n=10)  # Mengurangi ukuran populasi
ngen = 10  # Mengurangi jumlah generasi
cxpb = 0.5
mutpb = 0.3

result = algorithms.eaSimple(population, toolbox, cxpb, mutpb, ngen, stats=None, halloffame=None, verbose=False)

# Menggunakan fitur terbaik yang ditemukan oleh GA
best_individual = tools.selBest(population, 1)[0]
selected_features = [i for i, bit in enumerate(best_individual) if bit == 1]
X_best = X_normalized[:, selected_features]
```

Gambar 2. 9 Kode Optimasi *GA*

Berikut adalah Tabel 2.6 yang menyajikan daftar parameter yang digunakan dalam kode tersebut, beserta deskripsi fungsi dari masing-masing parameter:

Tabel 2. 6 Parameter Optimasi *GA*

Parameter	Keterangan
creator.create	Digunakan untuk mendefinisikan kelas baru yang mewakili individu dan fitness dalam populasi *GA*.
FitnessMax	Sebuah kelas yang mewakili jenis *fitness* yang ingin dimaksimalkan, dalam hal ini adalah akurasi.
Individual	Klas yang merepresentasikan setiap individu dalam populasi *GA*, berupa daftar fitur yang dipilih atau tidak.
evalRF	Fungsi evaluasi yang menghitung seberapa baik individu (set fitur) berdasarkan akurasi model *RandomForestClassifier*.
toolbox	Wadah untuk menyimpan fungsi yang digunakan dalam *GA* seperti inisialisasi, evaluasi, dan operasi genetik.
attr_bool	Fungsi untuk menginisialisasi atribut individu, dalam hal ini sebagai nilai biner (0 atau 1, fitur tidak dipilih atau dipilih).
individual	Fungsi untuk menginisialisasi sebuah individu berdasarkan attr_bool, menentukan panjang individu berdasarkan jumlah fitur.
population	Fungsi untuk menginisialisasi populasi yang terdiri dari individu-

	individu.
mate	Fungsi untuk melakukan persilangan (crossover) antar individu, menggunakan two-point crossover.
mutate	Fungsi untuk melakukan mutasi pada individu dengan probabilitas tertentu, menggunakan flip bit mutation.
select	Fungsi seleksi yang menentukan individu mana yang akan bertahan dan bereproduksi, menggunakan tournament selection.
evaluate	Fungsi evaluasi yang digunakan oleh GA untuk menilai individu.
eaSimple	Algoritma yang digunakan untuk menjalankan *GA*, yang mencakup proses seleksi, persilangan, dan mutasi secara berulang.
ngen	Parameter yang menentukan jumlah generasi yang akan dijalankan dalam simulasi *GA*.
cxpb	Probabilitas persilangan (crossover probability), mengontrol seberapa sering crossover terjadi antar individu.
mutpb	Probabilitas mutasi (mutation probability), mengontrol seberapa sering mutasi terjadi pada individu.
selBest	Memilih individu terbaik dari populasi setelah evolusi selesai.
selected_features	Daftar fitur yang dipilih oleh individu terbaik.
X_best	Dataset yang hanya berisi fitur-fitur yang dipilih oleh individu terbaik.

2.2.5 Evaluasi

Pada tahap evaluasi, penelitian ini akan mengukur keakuratan algoritma yang diterapkan dengan mempertimbangkan kualitas dari data latihan yang digunakan. Evaluasi ini akan dilakukan melalui pengujian menggunakan teknik *Confusion Matrix* untuk menentukan tingkat *Accuracy* (akurasi). Proses ini penting untuk memastikan bahwa algoritma yang digunakan dapat secara efektif mengklasifikasikan data dengan presisi yang tinggi.

Tabel 2.7 *Confusion Matrix*

Predicted	*Positive*	*Negative*
Actual Positive	*TP*	*FN*
Actual Negative	*FP*	*TN*

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \times 100\% \quad (2.2)$$

Berikut adalah penjelasan tentang istilah-istilah yang digunakan dalam Confusion Matrix:

TP (*True Positive*) adalah jumlah data yang benar-benar berlabel '*yes'* dan diidentifikasi secara benar.

TP (*True Negative*) adalah jumlah data yang benar-benar berlabel '*no*'dan diidentifikasi secara benar.

FP (*False Positive*) adalah jumlah data yang benar-benar berlabel '*yes'* dan diidentifikasi secara benar.

FN (*False Negative*) adalah jumlah data yang benar-benar berlabel '*no'* dan diidentifikasi secara benar.