BAB II

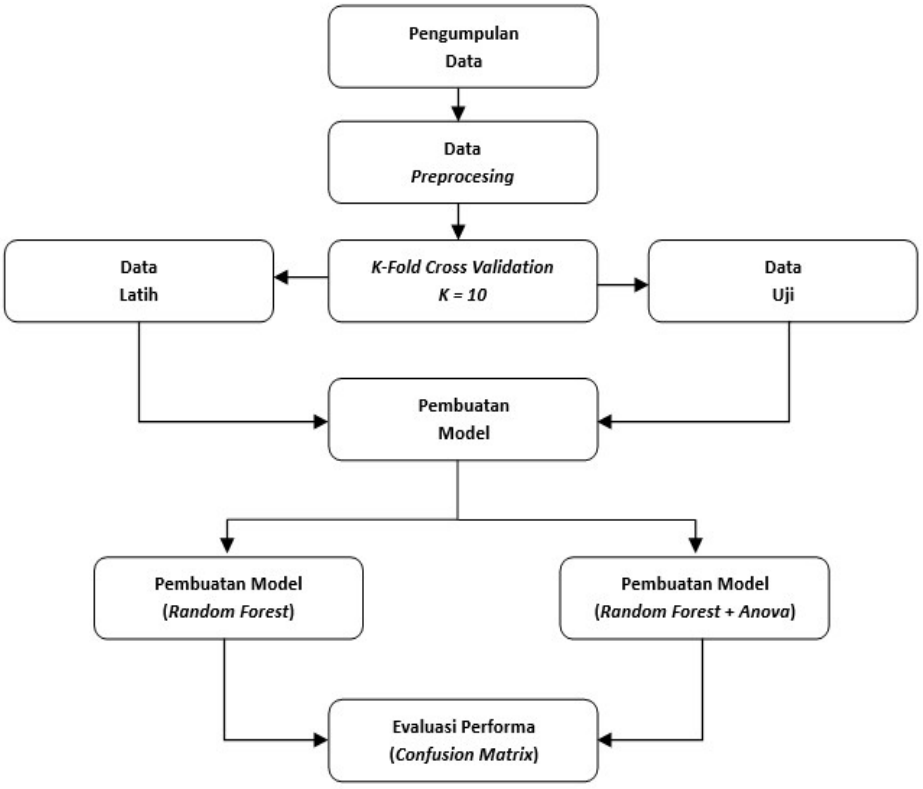
METODOLOGI PENELITIAN

2.1 Objek Penelitian

Stunting merupakan masalah kesehatan yang terjadi pada anak anak. Di kota Samarinda kasus Stunting terbilang cukup tinggi yaitu sebesar 25,3% (Cindy Mutia Annur, 2023). Oleh karena itu, Objek yang digunakan dalam penelitian ini adalah data Stunting pada tahun 2023 yang didapat dari 26 puskesmas di lingkup Dinas kesehatan Kota Samarinda. Penelitian dilakukan di Dinas Kesehatan Kota Samarinda yang berlokasi di Jl. Milono No.1, 75122, Bugis, Kec. Samarinda Kota, Kota Samarinda, Kalimantan Timur 76112.

2.2 Teknik Analisis Data

Penelitian ini bertujuan untuk mengembangkan sebuah metodologi yang terstruktur dan terorganisir dengan baik untuk memastikan bahwa proses penelitian dilaksanakan secara konsisten dan teratur. Teknik analisis data yang akan digunakan telah dirancang untuk mengikuti alur kerja yang sistematis, yang akan diuraikan dalam alur penelitian berikut :



Gambar 2. 1 Alur Penelitian

2.2.1 Pengumpulan data

Tahapan awal pada penelitian ini adalah melakukan perhaman pada data, dimulai dari pengumpulan data lalu dilanjutkan dengan proses data preparation atau persiapan data kasus Stunting. Data yang didapat memiliki 23 kolom dengan 22 atribut dan 1 atribut sebagai label (TB/U). Dapat dilihat pada Tabel 2.1 adalah data yang didapat dari Dinas kesehatan Kota Samarinda.

Tabel 2.1 Atribut Data Dinas Kesehatan Kota Samarinda

No	Atribut	Tipe Data	Keterangan
1	Nama	*String*	Nama
2	JK	*String*	Jenis Kelamin
3	BB Lahir	*Integer*	Berat Badan Lahir
4	TB Lahir	*Integer*	Tinggi Badan Lahir
5	Tgl Lahir	*Date*	Tanggal Lahir
6	Nama Ortu	*String*	Nama Orang Tua
7	Provinsi	*String*	Provinsi
8	Kab/Kota	*String*	Kabupaten atau Kota
9	Kec	*String*	Kecamatan
10	Puskesmas	*String*	Lokasi Puskesmas
11	Posyandu	*String*	Lokasi Posyandu
12	RT	*Integer*	Lokasi RT
13	Total Pengukuran	*Integer*	Total Pengukuran
14	Tanggal Pengukuran	*Integer*	Tanggal Pengukuran
15	Berat	*Integer*	Berat Badan
16	Tinggi	*Integer*	Tinggi Badan
17	BB/U	*numeric*	Berat Badan Menurut Umur
18	ZS BB/U	*numeric*	*Z Score* Berat Badan menurut Umur
19	TB/U	*numeric*	Tinggi Badan menurut Umur
20	ZS TB/U	*numeric*	*Z Score* Tinggi Badan menurut Umur
21	BB/TB	*numeric*	Berat Badan menurut Tinggi Badan
22	ZS BB/TB	*numeric*	*Z Score* Berat Badan menurut Tinggi Badan

2.2.2 Data *Pre-Procesing*

Pada Tahapan ini akan memiliki 4 tahapan, yaitu data *selection*, data *cleaning*, data *Transformation* dan data *Balancing*. Data yang digunakan berasal dari data Dinas Kesehatan Kota Samarinda dengan proses persiapan sebagai berikut :

a) Data *Selection*

Pada tahapan ini, akan dilakukan pengambilan data dengan memililih fitur atau atribut yang relevan untuk digunakan pada penelitian sedangkan fitur yang dianggap tidak relevan akan dihilangkan, seperti pada tabel 2.2 menampilkan data awal yang didapatkan dari Dinas Kesehatan Kota Samarinda memiliki 23 kolom, lalu didapatkan 11 kolom yang dianggap tidak relevan untuk digunakan dalam mengklasifikasikan stunting pada anak, setelah dilakukan proses seleksi maka didapatkan 13 atribut yang dijadikan fitur dan 1 atribut (TB/U) sebagai target atau kelas. Dapat dilihat pada Tabel 2.2 adalah data yang telah diseleksi.

Tabel 2.2 Data *Selection*

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

b) Data *Cleaning*

Pada Tahapan ini data yang digunakan akan dilakukan proses pembersihan dengan menghapus data yang memiliki nilai #N/A (*no value is available*) atau data yang tidak memiliki nilai serta yang kembar atau terduplikasi. Proses ini akan menggunakan bahasa pemrograman *Python* serta library Pandas, dengan fungsi ***dropna()*** untuk menghapus baris yang memiliki nilai hilang dan fungsi ***drop_duplicates()*** untuk menghapus baris yang merupakan duplikat.

```
missing_values = data.isna().sum()

# Menampilkan jumlah missing values untuk setiap kolom
print("Jumlah missing values untuk setiap kolom:")
print(missing_values)
```

Gambar 2. 2 Kode Menampilkan Nilai Kosong

Berikut adalah tabel 2.3 yang berisi tentang tiap parameter yang digunakan dalam kode tersebut beserta penjelasan fungsi tiap parameter.

Tabel 2.3 Parameter Kode Menampilkan Nilai Kosong

Parameter	Keterangan
data	Ini adalah variabel yang merujuk pada DataFrame pandas. DataFrame adalah struktur data dua dimensi yang bisa menyimpan data dalam bentuk tabel dengan baris dan kolom.
isna()	Fungsi isna() dari pandas digunakan untuk mendeteksi nilai-nilai yang hilang dalam DataFrame. Fungsi ini mengembalikan DataFrame baru di mana setiap entri adalah True jika entri asli adalah nilai yang hilang (NaN, None, atau NaT), dan False jika tidak.
sum()	Metode sum() digunakan di sini setelah isna() untuk menghitung jumlah True di setiap kolom dari DataFrame yang dihasilkan oleh isna(). Ini efektif menghitung jumlah nilai yang hilang dalam setiap kolom dari DataFrame asli data.

```python
# Menghitung jumlah data sebelum penghapusan duplikat
total_data_sebelum = len(data)

# 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')

#Menghapus Nilai Kosong Tiap Kolom
data.dropna(inplace=True)

# Menghitung jumlah data setelah penghapusan duplikat
total_data_sesudah = len(data)

print(f"Total data sebelum penghapusan duplikat: {total_data_sebelum}")
print(f"Total data setelah penghapusan duplikat: {total_data_sesudah}")
```

Gambar 2.3 Kode Menghapus Data Terduplikasi Dan Data Kosong

Berikut adalah tabel 2.4 yang berisi tentang tiap parameter yang digunakan dalam kode tersebut beserta penjelasan fungsi tiap parameter.

Tabel 2.4 Parameter Kode Menghapus Data Terduplikasi Dan Data Kosong

Fungsi		Parameter	Keterangan
len()	:	data	data adalah *DataFrame* yang sedang diproses. Fungsi *len*() di sini digunakan untuk menghitung jumlah baris dalam DataFrame.
sort_values()	:	*by*	Parameter by menerima list dari nama-nama kolom yang akan dijadikan dasar pengurutan. Di sini, pengurutan dilakukan berdasarkan kolom "Nama" dan "Tanggal Pengukuran".
		ascending	Parameter *ascending* menerima list dari nilai *boolean* yang menentukan urutan pengurutan untuk setiap kolom yang ditentukan di *by*. Nilai *True* untuk "Nama" berarti pengurutan secara *ascending* (A-Z), dan *False* untuk "Tanggal Pengukuran" berarti pengurutan secara *descending* (tanggal terbaru ke terlama).
drop_duplicates()	:	*subset*	Parameter *subset* menentukan kolom yang akan digunakan untuk identifikasi duplikat. Di sini, 'Nama' adalah kolom yang digunakan, yang berarti duplikat akan dicari berdasarkan nama.
		keep	Parameter *keep* menentukan duplikat mana yang akan dipertahankan. *first* menunjukkan bahwa hanya baris pertama dari setiap duplikat yang akan dipertahankan, yang di kasus ini adalah entri dengan 'Tanggal Pengukuran' terbaru karena pengurutan sebelumnya.
print()	:	*f-string*	*f-string* digunakan untuk memformat string, menyisipkan nilai dari variabel langsung ke dalam *string* yang dicetak. total_data_sebelum dan total_data_sesudah adalah variabel yang nilai-nya disisipkan dan dicetak.

c) Data *Transformation*

Pada tahapan ini dilakukan dengan proses konversi nilai-nilai atribut yang kategorikal menjadi bentuk numerik, tahapan ini perlu dilakukan karena dalam penggunaan library sklearn, hanya bisa menerima atribut dengan tipe numerik. Beberapa atribut yang ditransformasi meliputi jenis kelamin, naik berat badan, berat badan menurut umur, dan berat badan menurut tinggi badan. Proses ini akan menggunakan *library scikit-learn* dengan fungsi *LabelEncoder*. *Label Encoder* tersebut digunakan untuk mengubah teks atau data kategorikal menjadi data numerik dalam satu kolom data secara otomatis (M. M. Mafa'atiih, 2020)

```
from sklearn.preprocessing import LabelEncoder

# Buat instance LabelEncoder
encoder = LabelEncoder()

# Kolom yang perlu di-encode
columns_to_encode = ['JK', 'BB/U', 'BB/TB', 'Naik Berat Badan']

# Loop melalui setiap kolom yang perlu di-encode dan terapkan LabelEncoder
for column in columns_to_encode:
    data[column] = encoder.fit_transform(data[column])
```

Gambar 2.4 Kode Tranformasi Data Pada Kolom Atribut

```
# Ubah TB/U
data['TB/U'] = data['TB/U'].replace({'Normal': 0, 'Tinggi': 0, 'Pendek': 1, 'Sangat Pendek': 1})
```

Gambar 2.5 Kode Tranformasi Data Pada Kolom Label

Berikut adalah tabel 2.5 yang berisi tentang tiap parameter yang digunakan dalam kode tersebut beserta penjelasan fungsi tiap parameter.

Tabel 2.5 Parameter Kode Tranformasi

Fungsi	Parameter	Keterangan
LabelEncoder()	-	*LabelEncoder* adalah kelas dari *sklearn.preprocessing* yang digunakan untuk mengkonversi label kategori menjadi label numerik.
fit_transform()	*data[column]*	Metode *fit_transform()* digunakan untuk menyesuaikan encoder dan mengubah label menjadi bentuk *numerik*. Parameter *data[column]* merupakan kolom spesifik dari *DataFrame* data yang sedang diolah.
replace()	*{'Normal': 0, 'Tinggi': 0, 'Pendek': 1, 'Sangat Pendek': 1}*	Metode *replace()* digunakan untuk mengganti nilai-nilai dalam sebuah Series atau *DataFrame*. Parameter ini merupakan *dictionary* yang menentukan nilai lama dan baru; dalam hal ini, 'Normal' dan 'Tinggi' diganti dengan 0, 'Pendek' dan 'Sangat Pendek' diganti dengan 1.

d) Data *Balancing*

Pada tahapan terakhir preprocessing, ketidakseimbangan kelas akan terjadi ketika jumlah atribut mengalami ketidakseimbangan antara kelas mayoritas dan kelas minoritas. Sehingga Perlu melakukan data balancing menggunakan modul *python imblearn.over_sampling* dengan mengimpor fungsi *SMOTE* (*sampling strategy*) untuk melakukan *oversampling*.

```
x = data.drop(['TB/U'],axis=1)
y = data['TB/U']
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=42, sampling_strategy=1)
X_imb, y_imb = sm.fit_resample(x, y)
```

Gambar 2.6 Kode Data *Balancing SMOTE*

Berikut adalah tabel 2.6 yang berisi tentang tiap parameter yang digunakan dalam kode tersebut beserta penjelasan fungsi tiap parameter.

Tabel 2.6 Parameter Kode Data *Balancing SMOTE*

Fungsi	Parameter	Keterangan
drop()	*['TB/U']*	Parameter ini menentukan nama kolom atau daftar kolom yang akan dihapus dari *DataFrame* data. Di sini, kolom 'TB/U' akan dihapus.
	axis=1	*Axis* 1 mereferensikan kolom. Dengan *axis*=1, operasi *drop*() akan menghapus kolom yang ditentukan.
SMOTE	*random_state=42*	Memastikan bahwa hasil *oversampling* konsisten di berbagai eksekusi dengan menyediakan *seed* untuk generator bilangan acak. 42 adalah nilai yang sering digunakan dalam contoh untuk tujuan *reproducibilitas*.
	sampling_strategy=1	Menentukan strategi *sampling* yang diinginkan. Nilai 1 mengindikasikan bahwa kelas minoritas akan di*resample* hingga jumlahnya sama dengan kelas mayoritas.
fit_resample()	*x, y*	Metode *fit_resample()* digunakan untuk mengajarkan model *SMOTE* pada data input (x) dan target (y), kemudian mengembalikan dataset yang telah di-resample. x adalah *DataFrame* yang dihasilkan setelah menghapus kolom 'TB/U', dan y adalah nilai target yang berasal dari kolom 'TB/U'.

2.2.3 Pembagian data

Sebelum memulai proses pemodelan, dataset yang akan digunakan harus dibagi menjadi dua bagian utama, yaitu data latih dan data uji. Data latih berfungsi sebagai dasar untuk pembangunan model, sementara data uji digunakan untuk mengevaluasi kinerja model yang telah dibuat. Dalam penelitian ini, akan menggunakan teknik *K-Fold Cross Validation*, Hal ini dapat membantu mengidentifikasi

apakah model memiliki tingkat akurasi yang baik, serta membantu menghindari *overfitting* atau *underfitting* (Hafid, 2023). *K-Fold Cross Validation* akan diimplementasikan melalui *library* sklearn.model_selection dengan fungsi cross_val_score pada *Python*. Teknik ini melibatkan pembagian data menjadi 10 segmen atau kelompok, sesuai dengan parameter Cv yang ditetapkan sebesar 10, yang berarti data akan dibagi menjadi 10 bagian. Setiap segmen akan secara bergiliran berperan sebagai data latih dan data uji. Pada Penelitian ini menggunakan nilai k sebesar 10, yang menandakan bahwa eksperimen akan dijalankan sebanyak 10 kali. Setelah melakukan 10 kali percobaan, maka nilai rata-rata dari semua hasil pengujian akan diambil sebagai indikator untuk mendapatkan hasil yang lebih presisi dan akurat.

Tabel 2.7 *K-fold Cross Validation*

1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10

Pada Tabel 2.7, skema 10-fold CV dijelaskan, di mana setiap baris mewakili satu iterasi. Blok berwarna merah menunjukkan data yang digunakan sebagai data uji dalam iterasi tersebut, sedangkan blok berwarna putih menunjukkan data latih. Contoh visualisasinya adalah sebagai berikut, Pada Iterasi 1, Fold 1 digunakan sebagai data uji dan Fold 2 hingga Fold 10 sebagai data latih, pada Iterasi 2, Fold 2 sebagai data uji dan Fold 1 serta Fold 3 hingga Fold 10 sebagai data latih. Demikian seterusnya hingga Iterasi 10, di mana Fold 10 digunakan sebagai data uji dan Fold 1 hingga Fold 9 sebagai data latih. Teknik ini memastikan bahwa setiap fold digunakan sebagai data uji sekali, dan data latih pada iterasi lainnya, untuk memberikan evaluasi model yang lebih akurat dan menghindari overfitting.

2.2.4 Pembuatan Model

Berikut ini merupakan gambaran dan penjelasan dari model klasifikasi *Random Forest* yang digunakan pada penelitian in :

a) Dalam fase ini, dataset akan dilakukan pembagian menjadi dua segmen utama: data latih dan data uji. Data latih berperan sebagai pondasi untuk proses pembentukan model, sedangkan data uji digunakan sebagai alat untuk mengevaluasi efektivitas model yang sudah dibangun. Penelitian ini menggunakan metode *K-Fold Cross Validation*, sebuah teknik yang diaktualisasikan melalui penggunaan *library* sklearn.model_selection dan fungsi *cross_val_score* pada bahasa pemrograman *Python*. Dengan parameter Cv yang ditetapkan sebesar 10, ini mengindikasikan bahwa dataset akan terbagi menjadi 10 bagian atau kelompok yang sama. adapun model yang dipilih untuk melakukan klasifikasi Stunting di Kota Samarinda adalah model *Random Forest*. Berikut adalah penjelasan mengenai mekanisme kerja dari metode tersebut:

$$l(\mathrm{y}) = argmax_c(\sum_{n=1}^{N} I_{hn}(\mathrm{y}) = C) \quad (2.1)$$

a. Langkah Pertama sampel dataset tersebut dimanfaatkan untuk membentuk pohon keputusan atau *tree* ke-i (i = 1, 2, ...k). Dalam tahap pembuatan tree ini, digunakan metode *Classification and Regression Tree* (CART), di mana gini impurity diaplikasikan sebagai kriteria untuk menentukan pembagian pada setiap node dalam tree (Yoga Siswa, 2023) .

```
#Kode Inisialisasi Random Forets
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(max_depth=1, random_state=42)
```

Gambar 2.7 Kode Inisialisasi Model *Random Forest*

b. Langkah kedua melakukan pembagian data menggunakan *K-Fold Cross Validation* dengan K = 10.

```
# Kode kfold Cross Validation
from sklearn.model_selection import cross_val_score, StratifiedKFold
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
```

Gambar 2.8 Kode Pembagian Data Latih dan Data Uji

Berikut adalah tabel 2.8 yang berisi tentang tiap parameter yang digunakan dalam kode tersebut beserta penjelasan fungsi tiap parameter:

Tabel 2.8 Parameter Pembagian Data Latih dan Data Uji

Parameter	Keterangan
n_splits	Jumlah lipatan (fold) yang akan dibuat. Dalam kasus ini, nilai n_splits = 10 berarti akan memiliki 10 lipatan untuk cross-validation.
shuffle	Menentukan apakah data akan diacak sebelum dibagi menjadi lipatan. Jika shufffle = True, data akan diacak sebelum dibagi, sehingga setiap lipatan akan memiliki sampel yang berbeda-beda. Jika shufffle = false, data akan dibagi berdasarkan urutan aslinya.
random_state	Menentukan seed untuk generator nomor acak yang digunakan saat mengacak data. Penggunaan nilai random_state yang sama akan menghasilkan pembagian yang sama saat melakukan pengacakan.

c. Langkah ketiga dalam pemodelan algoritma Random Forest melibatkan penggunaan *library scikit-learn* untuk mengimplementasikan model. Pada tahap ini, akan menggunakan metode *k-fold cross-validation* untuk membagi dataset menjadi data latih dan uji, serta untuk mengukur performa model secara objektif. Pertama, membuat model *Random Forest* menggunakan *RandomForestClassifier* dari *scikit-learn*. Selanjutnya, melakukan perulangan melalui setiap *fold* yang dihasilkan oleh *k-fold cross-validation*. Pada setiap perulangan, dataset dibagi menjadi data latih dan uji , dan model *Random Forest* dilatih menggunakan data latih. Setelah dilatih, model tersebut digunakan untuk melakukan prediksi pada data uji.

Skor evaluasi akurasi dihitung menggunakan metrik yang sesuai dari library *sklearn.metrics* dan disimpan untuk setiap *fold*. Selain itu, matriks kebingungan (confusion matrix) dari setiap *fold* juga dihitung menggunakan fungsi *confusion_matrix* dari *scikit-learn*. Hasil prediksi dan skor evaluasi ini kemudian disimpan untuk digunakan dalam analisis lebih lanjut tentang performa model pada dataset yang diberikan.

```
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
import numpy as np

# Perform cross-validation on the Random Forest classifier with K=10
clf = RandomForestClassifier(max_depth=1, random_state=42)

# Kode kfold Cross Validation
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Lists to store evaluation metrics for each fold
conf_matrices = []
accuracies = []
precisions = []
recalls = []
f1_scores = []

for train_index, test_index in cv.split(X_imb, y_imb):
    X_train, X_test = X_imb.iloc[train_index], X_imb.iloc[test_index]
    y_train, y_test = y_imb.iloc[train_index], y_imb.iloc[test_index]

    # Train the model
    clf.fit(X_train, y_train)

    # Predict on the test set
    y_pred = clf.predict(X_test)

    # Calculate evaluation metrics
    conf_matrices.append(confusion_matrix(y_test, y_pred))
    accuracies.append(accuracy_score(y_test, y_pred))
```

Gambar 2.9 Kode Permodelan Algoritma *Random Forest*

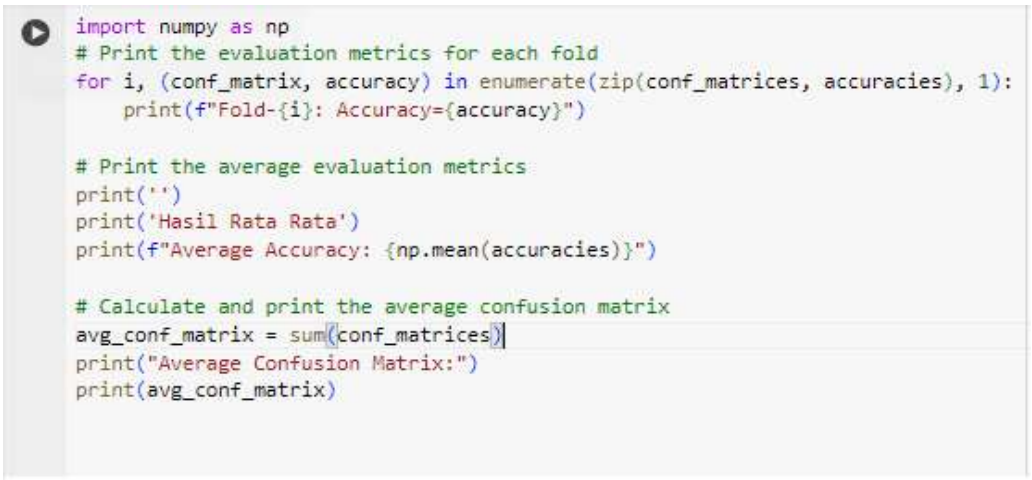
Berikut adalah tabel 2.9 yang berisi tentang tiap parameter yang digunakan dalam kode tersebut beserta penjelasan fungsi tiap parameter:

Tabel 2.9 Parameter Permodelan Algoritma *Random Forest*

Parameter	Keterangan
clf	Model *Random Forest* yang digunakan untuk melakukan klasifikasi.
train_index	Indeks data untuk data latih.
test_index	Indeks data untuk data uji.
accuracies	List yang akan menyimpan skor akurasi dari setiap *fold*.
confusion_matrices	List yang akan menyimpan *confusion matrix* dari setiap *fold*.
X_train	Subset atribut untuk training.
X_test	Subset atribut untuk testing.
y_train	Subset label untuk training..
y_test	Subset label untuk testing.
y_pred	Hasil prediksi dari subset test.

d. Langkah keempat Setelah proses *K-fold cross-validation* selesai dilakukan dan model *Random Forest* telah diuji pada setiap *fold*, skor evaluasi seperti akurasi dari setiap *fold* dapat ditampilkan. Dalam pengulangan menggunakan *enumerate* dan *zip*, akurasi dari masing-masing *fold* diproses dan ditsmpilkan. Selain itu, rata-rata akurasi dari semua *fold* juga dihitung dengan menggunakan fungsi *np.mean()* dari *NumPy*. Hasil rata-rata skor evaluasi akurasi dari seluruh *fold* juga dicetak. Terakhir, rata-rata dari *confusion matrix* dari semua *fold*

dihitung menggunakan fungsi *np.mean()* dan ditampilkan. *Confusion matrix*) rata-rata ini memberikan gambaran tentang seberapa baik model dapat mengklasifikasikan setiap kelas pada data uji.



```python
import numpy as np
# Print the evaluation metrics for each fold
for i, (conf_matrix, accuracy) in enumerate(zip(conf_matrices, accuracies), 1):
    print(f"Fold-{i}: Accuracy={accuracy}")

# Print the average evaluation metrics
print('')
print('Hasil Rata Rata')
print(f"Average Accuracy: {np.mean(accuracies)}")

# Calculate and print the average confusion matrix
avg_conf_matrix = sum(conf_matrices)
print("Average Confusion Matrix:")
print(avg_conf_matrix)
```

Gambar 2. 10 Kode Hasil akurasi dari setiap *Fold*

Berikut adalah tabel 2.10 yang berisi tentang tiap parameter yang digunakan dalam kode tersebut beserta penjelasan fungsi tiap parameter

Tabel 2. 10 Parameter Hasil Skor Dari Setiap *Fold*

Parameter	Keterangan
np	Alias untuk *numpy*, sebuah library *Python* yang digunakan untuk melakukan operasi numerik pada *array*, matriks, dan struktur data numerik lainnya.
enumerate	Fungsi *Python* yang digunakan untuk mengambil elemen dari sebuah iterable (seperti *list, tuple,* atau *string*) bersamaan dengan indeksnya dalam bentuk *tuple*.
zip	Fungsi *Python* yang digunakan untuk menggabungkan elemen-elemen yang bersesuaian dari dua atau lebih *iterable* (*list, tuple,* atau *array*) menjadi *tuple-tuple* baru
accuracies	List yang menyimpan nilai akurasi untuk setiap *fold* dalam *cross-validation*
mean	Fungsi numpy yang digunakan untuk menghitung rata-rata dari sebuah array atau list
axis	Parameter yang digunakan dalam fungsi *np.mean* untuk menentukan sumbu (axis) mana yang akan dihitung rata-ratanya
avg_cm	Nilai rata-rata dari *confusion matrix* untuk semua *fold* dalam *cross-validation*

b) Tahap seleksi fitur menggunak metode *ANOVA* dengan cara menentukan atribut yang memiliki bobot atau pengaruh signifikan terhadap variabel yang akan diklasifikasi. Nilai F yang lebih besar menunjukkan bahwa ada perbedaan yang signifikan antara kelompok, karena menunjukkan bahwa variansi antar kelompok lebih besar dibandingkan dengan variansi dalam kelompok. Sebaliknya,

nilai F yang rendah menunjukkan bahwa perbedaan antar kelompok tidak signifikan, menunjukkan bahwa kelompok-kelompok tersebut memiliki variansi yang serupa. Berikut tahapan kerja dari seleksi fitur *Analysis of Variance* (ANOVA) adalah sebagai berikut :

a. Semua fitur dipilih pada dataset Stunting
b. Fungsi fitur target dari scikit-learn dihitung menggunakan *ANOVA F-Score* untuk setiap fitur.Berikut adalah rumus untuk menghitung *ANOVA*:

$$F = \frac{variance\ between\ groups}{variance\ within\ groups} \tag{2.2}$$

$$Variance\ between\ groups = \frac{\sum_i^n n_i\,(\bar{Y}_{i\,-}\,\bar{Y})^2}{(k-1)} \tag{2.3}$$

$$Variance\ within\ groups = = \frac{\sum_{i=1}^{k}\sum_{j=1}^{n_i}(\bar{Y}_{ij\,-}\,\bar{Y})^2}{(n-k)} \tag{2.4}$$

Keterangan :

n_i = Jumlah sampel di grup ke-i
$\bar{Y}_i$ = Rata-rata sampel di grup ke-i
$\bar{Y}$ = Rata-rata total sampe
k = Jumlah grup
n = Jumlah total sampel

c. Hasil dari pengujian akan digunakan dalam pemilihan fitur yang mempunyai pengaruh tinggi dan varian terendah dipilih dalam percobaan ini lalu akan diuji dengan SelectKbeast(); k mewakili jumlah fitur yang digunakan untuk dataset akhir
d. Jumlah fitur dengan pengaruh tertinggi akan digunakan untuk membuat berbagai subset fitur.

```
from sklearn.feature_selection import SelectKBest, f_classif

X = data

# Menggunakan SelectKBest dengan ANOVA untuk memilih fitur terbaik
selector = SelectKBest(score_func=f_classif)
X_selected = selector.fit_transform(X, y)

# Mengambil nilai F dan nama fitur dari SelectKBest
f_values = selector.scores_
features = X.columns

# Membuat DataFrame dari fitur dan nilai F
import pandas as pd
df_f = pd.DataFrame({'Feature': features, 'F_Value': f_values})

# Sorting DataFrame berdasarkan nilai F secara ascending
df_sorted = df_f.sort_values('F_Value', ascending=False)

# Membuat diagram bar untuk menampilkan nilai F dari semua fitur
plt.figure(figsize=(12, 6))
plt.bar(df_sorted['Feature'], df_sorted['F_Value'], color='skyblue')
plt.xlabel('Fitur')
plt.ylabel('Nilai F')
plt.title('Perbandingan Nilai F dari Semua Fitur (Ascending)')
plt.xticks(rotation=90)
plt.show()
```

Gambar 2. 11 Kode Seleksi Fitur *ANOVA*

Berikut adalah tabel 2.11 yang berisi tentang tiap parameter yang digunakan dalam kode tersebut beserta penjelasan fungsi tiap parameter

Tabel 2.11 Parameter Seleksi Fitur *ANOVA*

Fungsi	Parameter	Keterangan
SelectKBest	*score_func=f_classif*	*SelectKBest* adalah metode untuk seleksi fitur yang memilih K fitur terbaik berdasarkan nilai statistik tertentu. *score_func = f_classif* menunjukkan penggunaan *F-test ANOVA* untuk tugas klasifikasi, yang mengukur pengaruh tiap fitur terhadap *variabilitas* antara kelas.
fit_transform()	*X, y*	Metode *fit_transform()* digunakan untuk mengajarkan *selector* pada data X (fitur) dan y (target) dan mengubah X menjadi subset fitur yang terpilih berdasarkan skor tertinggi.
DataFrame()	*{'Feature': features, 'F_Value': f_values}*	Membuat *DataFrame* dari dua *arrays,* features yang berisi nama fitur dari X dan *f_values* yang berisi nilai F dari setiap fitur setelah dihitung oleh *SelectKBest.*
sort_values()	*'F_Value', ascending=False*	Mengurutkan DataFrame berdasarkan kolom *F_Value* secara descending untuk melihat fitur dengan pengaruh paling tinggi terlebih dahulu.

2.2.5 Evaluasi

Ditahapan evaluasi akan dilakukan pengukuran akurasi dari algoritma yang digunakan dengan kualitas data training serta akan diuji dengan menggunakan teknik C*onfusion Matrix.*

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \times 100\% \quad (2.5)$$

Keterangan :

TP (*True Positive*) : Jumlah data point berlabel *yes* yang nilainya diidentifikasi benar.
TP (True Negative) : Jumlah data point berlabel no yang nilainya diidentifikasi salah.
FP (False Positive) : Jumlah data point berlabel yes yang nilai sebenarnya diidentifikasi salah
FN (False Negative) : Jumlah data point berlabel no yang nilai sebenarnya teridentifikasi benar