DIGITAL SKILL FAIR DATA SCIENCE 35.0

Stunting Detection in Toddlers Using Support Vector Machine (SVM) Classification

Rahfa Qur'aniyatin Dhuha

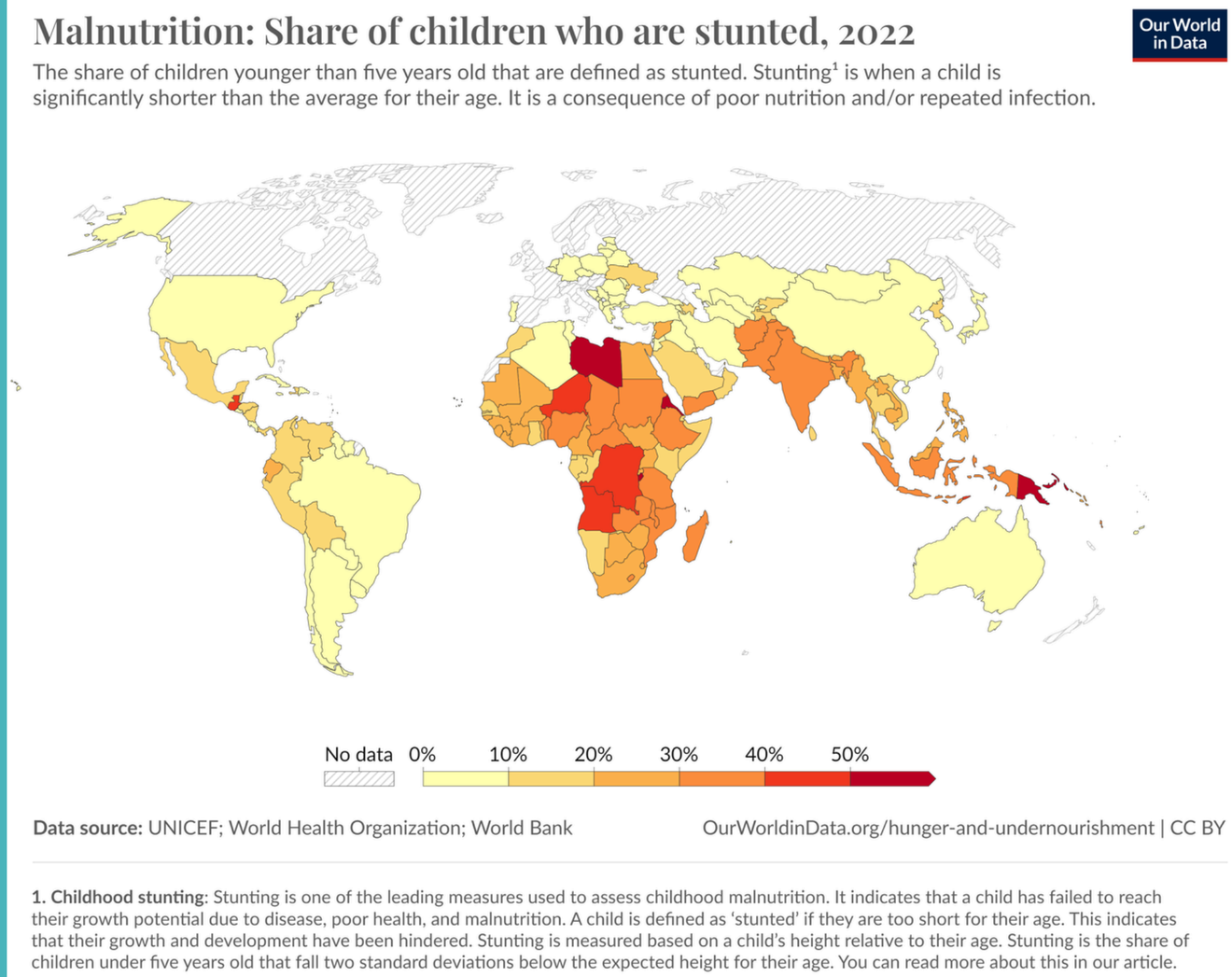
rahfaqurniya@gmail.com

www.linkedin.com/in/rahfaquraniyatin

What is stunting ?

Stunting is the impaired growth and development that children experience from poor nutrition, repeated infection, and inadequate psychosocial stimulation. Children are defined as stunted if their height-for-age is more than two standard deviations below the WHO Child Growth Standards median.

— World Health Organization (WHO)



Source:*https://ourworldindata.org/grapher/share-of-children-younger-than-5-who-suffer-from-stunting*

kaggle

Search

Create

Home

Competitions

Datasets

Stunting Toddler (Balita) Detection (121K rows)

Stunting Toddler (Under 5 Years Childern) Detection with 121K rows dataset

Dataset

This "Stunting Baby/Toddler Detection" dataset is based on the z-score formula for determining stunting according to WHO (World Health Organization), that focuses on stunting detection in children under five years old. It consists of 121,000 rows of data, detailing information on the age, sex, height, and nutritional status of toddlers.

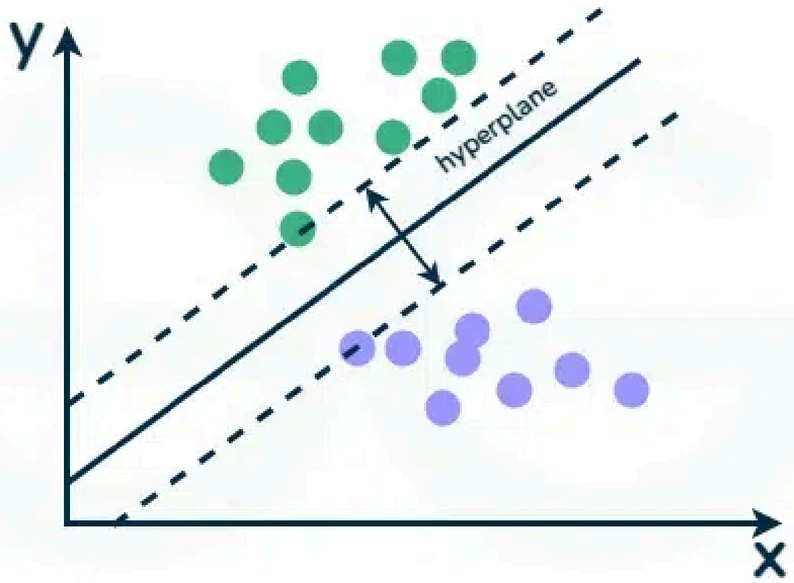
	Umur (bulan)	Jenis Kelamin	Tinggi Badan (cm)	Status Gizi
0	0	laki-laki	44.591973	stunted
1	0	laki-laki	56.705203	tinggi
2	0	laki-laki	46.863358	normal
3	0	laki-laki	47.508026	normal
4	0	laki-laki	42.743494	severely stunted
...	...	...	...	...
120994	60	perempuan	100.600000	normal
120995	60	perempuan	98.300000	stunted
120996	60	perempuan	121.300000	normal
120997	60	perempuan	112.200000	normal
120998	60	perempuan	109.800000	normal

120999 rows × 4 columns

Source:*https://www.kaggle.com/datasets/rendiputra/stunting-balita-detection-121k-rows/data*

ibimbing

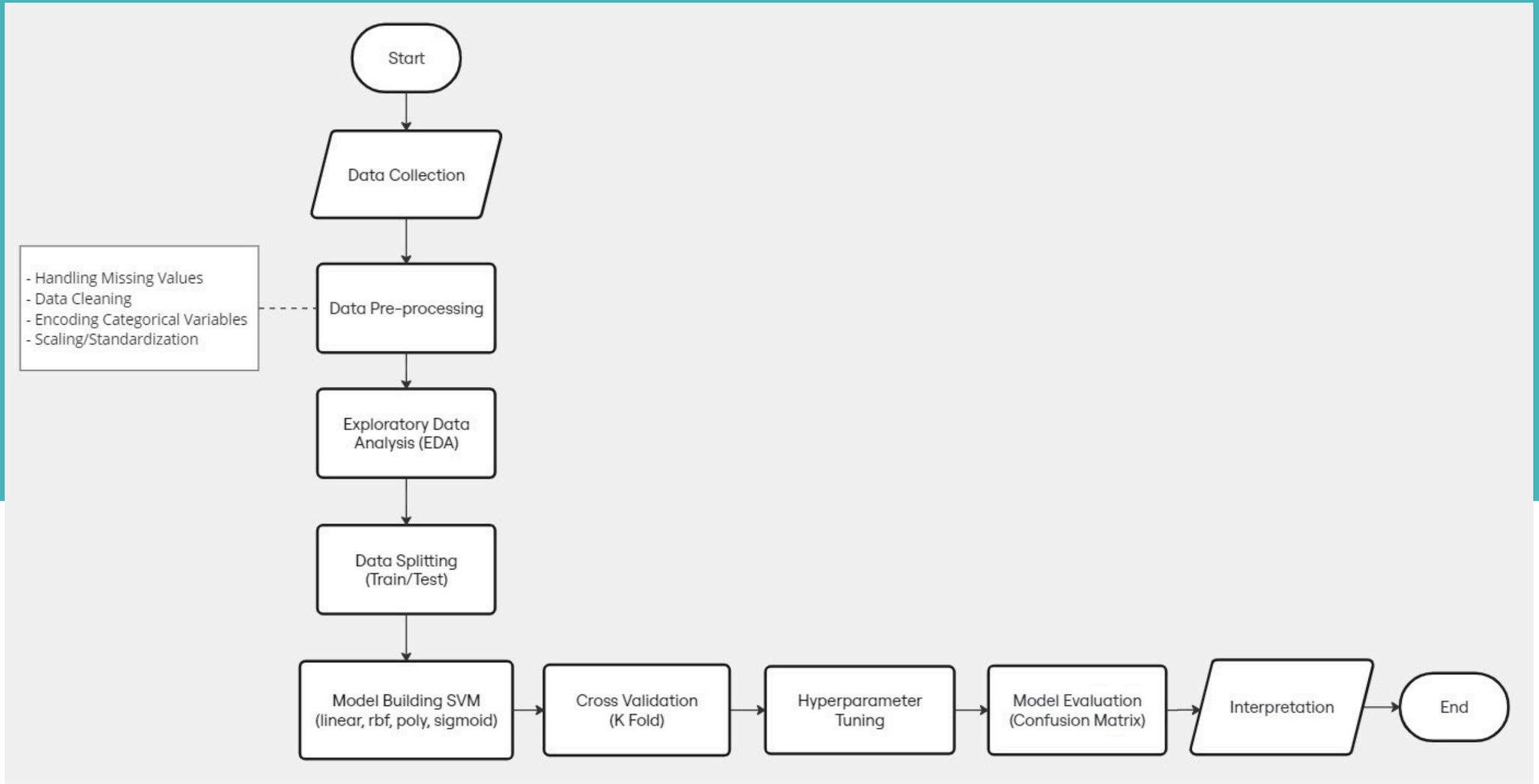
Support Vector Machine (SVM)



A Support Vector Machine (SVM) is a supervised machine learning algorithm used for both classification and regression tasks. While it can be applied to regression problems, SVM is best suited for classification tasks. The primary objective of the SVM algorithm is to identify the optimal hyperplane in an N-dimensional space that can effectively separate data points into different classes in the feature space. The algorithm ensures that the margin between the closest points of different classes, known as support vectors, is maximized.

Source: *www.geeksforgeeks.org/support-vector-machine-algorithm/*

Flowchart


Start
Data Collection
- Handling Missing Values
- Data Cleaning
- Encoding Categorical Variables
- Scaling/Standardization
Data Pre-processing
Exploratory Data Analysis (EDA)
Data Splitting (Train/Test)
Model Building SVM (linear, rbf, poly, sigmoid)
Cross Validation (K Fold)
Hyperparameter Tuning
Model Evaluation (Confusion Matrix)
Interpretation
End

ibimbing

Implementing SVM Algorithm in Python

Import Data

```python
# import data
import pandas as pd

stunting_data = pd.read_csv('/content/data_balita.csv')
stunting_data.head()
```

	Umur (bulan)	Jenis Kelamin	Tinggi Badan (cm)	Status Gizi
0	0	laki-laki	44.591973	stunted
1	0	laki-laki	56.705203	tinggi
2	0	laki-laki	46.863358	normal
3	0	laki-laki	47.508026	normal
4	0	laki-laki	42.743494	severely stunted

Dataset Column Details:

- **Age (Month):** Indicates the age of a baby/toddler in months (ages 0 to 60 months).
- **Gender:** There are two categories in this column, 'male' and 'female'.
- **Height:** Recorded in centimeters, height is a key indicator for assessing the physical growth of children under five.
- **Nutrition Status:** This column is categorized into 4 statuses - 'severely stunted', 'stunted', 'normal', and 'tall'. 'Severely stunted' indicates a very serious condition (<-3 SD), 'stunted' indicates a stunted condition (-3 SD to <-2 SD), 'normal' indicates a healthy nutritional status (-2 SD to +3 SD), and 'tall' indicates above-average growth (>+3 SD).

ibimbing

Pre-Processing Data

Rename Columns

```python
# rename coloumns
stunting_data = stunting_data.rename(columns={"Umur (bulan)":"Umur","Tinggi Badan (cm)":"Tinggi Badan"})
print(stunting_data)
```

```
        Umur Jenis Kelamin  Tinggi Badan       Status Gizi
0          0     laki-laki     44.591973           stunted
1          0     laki-laki     56.705203            tinggi
2          0     laki-laki     46.863358            normal
3          0     laki-laki     47.508026            normal
4          0     laki-laki     42.743494  severely stunted
...      ...           ...           ...               ...
120994    60     perempuan    100.600000            normal
120995    60     perempuan     98.300000           stunted
120996    60     perempuan    121.300000            normal
120997    60     perempuan    112.200000            normal
120998    60     perempuan    109.800000            normal

[120999 rows x 4 columns]
```

Pre-Processing Data

Check the data info

```
# check the data info
stunting_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 120999 entries, 0 to 120998
Data columns (total 4 columns):
 #   Column          Non-Null Count   Dtype  
---  ------          --------------   -----  
 0   Umur            120999 non-null  int64  
 1   Jenis Kelamin   120999 non-null  object 
 2   Tinggi Badan    120999 non-null  float64
 3   Status Gizi     120999 non-null  object 
dtypes: float64(1), int64(1), object(2)
memory usage: 3.7+ MB
```

As we can see from the above info that the our dataset has 4 columns and each columns has 120999 values. There is no Null values in the dataset.

We can also check the null values using df.isnull()

```
# check the missing value
missing_values = stunting_data.isnull().sum()
print("Number of missing values:")
print(missing_values)
```

```
Number of missing values:
Umur              0
Jenis Kelamin     0
Tinggi Badan      0
Status Gizi       0
dtype: int64
```

Pre-Processing Data

Check the duplicates

```
# check the duplicates
duplicates = stunting_data[stunting_data.duplicated()]
print("Duplicate rows :")
print(duplicates)
```

```
Duplicate rows :
        Umur Jenis Kelamin  Tinggi Badan       Status Gizi
6012       3     laki-laki          62.1            normal
6014       3     laki-laki          59.2            normal
6027       3     laki-laki          61.4            normal
6031       3     laki-laki          71.0            tinggi
6032       3     laki-laki          51.5  severely stunted
...      ...           ...           ...               ...
120994    60     perempuan         100.6            normal
120995    60     perempuan          98.3           stunted
120996    60     perempuan         121.3            normal
120997    60     perempuan         112.2            normal
120998    60     perempuan         109.8            normal

[81574 rows x 4 columns]
```

The dataset contains 81,574 duplicate rows.

Remove the duplicates

```
# removing duplicates
stunting_data1 = stunting_data.drop_duplicates()

# rechecking for duplicated data
stunting_data1.duplicated().sum()
```

```
0
```

Pre-Processing Data

Check the outlier

```python
# check the outlier

def check_outliers_iqr(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = df[(df[column] < lower_bound) | (df[column] > upper_bound)]
    return outliers

# Tinggi Badan column:
outliers_tinggi_badan = check_outliers_iqr(stunting_data1, 'Tinggi Badan')
print("Outliers in 'Tinggi Badan':")
print(outliers_tinggi_badan)

# Umur column:
outliers_umur = check_outliers_iqr(stunting_data1, 'Umur')
print("Outliers in 'Umur':")
print(outliers_umur)
```

```
Outliers in 'Tinggi Badan':
Empty DataFrame
Columns: [Umur, Jenis Kelamin, Tinggi Badan, Status Gizi]
Index: []
Outliers in 'Umur':
Empty DataFrame
Columns: [Umur, Jenis Kelamin, Tinggi Badan, Status Gizi]
Index: []
```

The output shows that no outliers were detected based on the IQR method.

Label Encoding

(for categorical columns)

```python
# Label Encoding (for categorical columns)
def label_encoding(df, column, mapping):
    df[column] = df[column].map(mapping)
    return df

gender_mapping = {'laki-laki': 0, 'perempuan': 1}
status_gizi_mapping = {'severely stunted': 0, 'stunted': 1, 'normal': 2, 'tinggi': 3}

stunting_data1 = label_encoding(stunting_data1, 'Jenis Kelamin', gender_mapping)
stunting_data1 = label_encoding(stunting_data1, 'Status Gizi', status_gizi_mapping)

print(stunting_data1.head())
```

```
   Umur  Jenis Kelamin  Tinggi Badan  Status Gizi
0     0              0     44.591973            1
1     0              0     56.705203            3
2     0              0     46.863358            2
3     0              0     47.508026            2
4     0              0     42.743494            0
<ipython-input-88-9327fd32636d>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

Pre-Processing Data

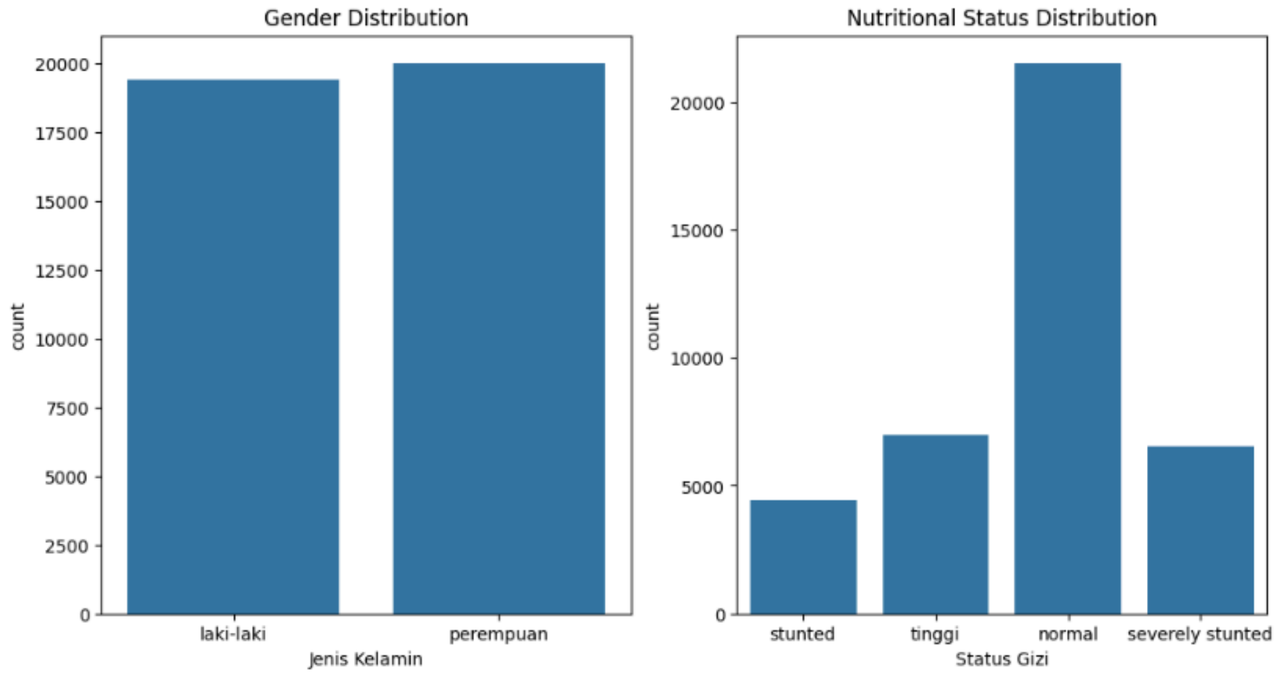
Standardization

```python
# Standardization
from sklearn import preprocessing

# Initialize the StandardScaler
scaler = preprocessing.StandardScaler()

# Select numerical columns for scaling (exclude 'Status Gizi' and 'Jenis Kelamin')
numerical_cols = ['Umur', 'Tinggi Badan']

# Fit and transform the numerical columns
stunting_data1[numerical_cols] = scaler.fit_transform(stunting_data1[numerical_cols])

print(stunting_data1.head())
```

```
      Umur  Jenis Kelamin  Tinggi Badan  Status Gizi
0 -1.469424              0     -2.096915            1
1 -1.469424              0     -1.484093            3
2 -1.469424              0     -1.982003            2
3 -1.469424              0     -1.949388            2
4 -1.469424              0     -2.190431            0
<ipython-input-27-cca0f0bc9202>:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

- Standardization is a useful technique to transform attributes with a Gaussian distribution and differing means and standard deviations to a standard Gaussian distribution with a mean of 0 and a standard deviation of 1.
- We can standardize data using scikit-learn with the StandardScaler class.
- It works well when the features have a normal distribution or when the algorithm being used is not sensitive to the scale of the features.

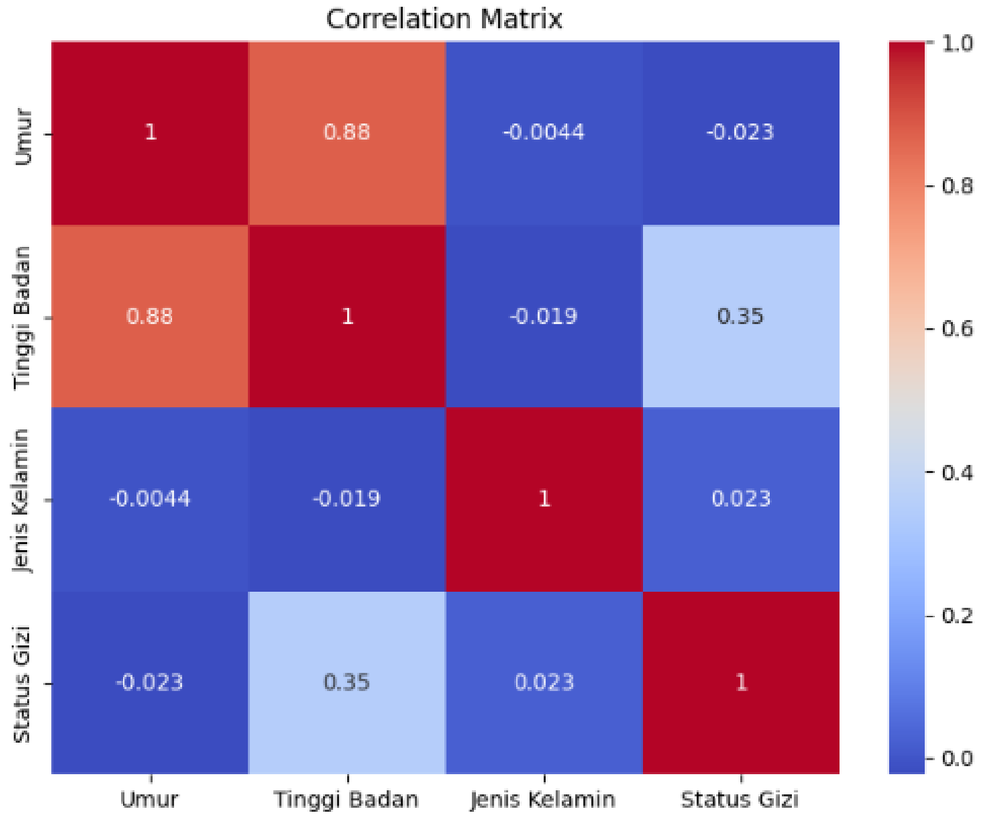
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Count plots for categorical variable

# Gender distribution
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
sns.countplot(x='Jenis Kelamin', data=stunting_data1)
plt.title('Gender Distribution')

# Nutritional Status Distribution
plt.subplot(1, 2, 2)
sns.countplot(x='Status Gizi', data=stunting_data1)
plt.title('Nutritional Status Distribution')
plt.show()
```



Exploratory Data Analysis (EDA)

Bar plot

- **Gender Distribution:** this chart shows a nearly equal distribution between male ("laki-laki") and female ("perempuan") participants, each with approximately 20,000 individuals.
- **Nutritional Status Distribution:** the "normal" category has the highest count, followed by "tinggi" (high) and "severely stunted," with "stunted" being the least frequent category.



Exploratory Data Analysis (EDA)

Correlation Matrix

```python
# Correlation matrix
correlation_matrix = stunting_data1[['Umur', 'Tinggi Badan',
                                     'Jenis Kelamin', 'Status Gizi']].corr()
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

- There is a strong positive correlation (0.88) between age (Umur) and height (Tinggi Badan), indicating that height tends to increase with age.
- A moderate positive correlation (0.35) exists between height (Tinggi Badan) and nutritional status (Status Gizi), suggesting better nutritional status is associated with greater height.
- Gender (Jenis Kelamin) shows very weak or negligible correlations with other variables, with values close to zero.
- Overall, significant relationships are observed between age, height, and nutritional status, while gender has minimal influence.

Modelling and Evaluation

Splitting the Data

```
[114] # Declare feature vector and target variable
      X = stunting_data1.drop(['Status Gizi'], axis=1)
      y = stunting_data1['Status Gizi']
```

```
[115] # split X and y into training and testing sets
      from sklearn.model_selection import train_test_split

      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
[116] # check the shape of X_train and X_test
      X_train.shape, X_test.shape
```

```
((31540, 3), (7885, 3))
```

The data was split into an 80:20 ratio, where 80% (31,540 samples) is used for training and 20% (7,885 samples) is used for testing. Both the training and testing sets maintain the same number of features (3), ensuring consistency in feature representation between the two sets.

Modelling and Evaluation

Support Vector Machine (SVM)

```python
# Function to perform SVM with different kernels
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report


def perform_svm(X_train, X_test, y_train, y_test, kernel='linear', C=1, gamma='scale'):
    svm_classifier = SVC(kernel=kernel, C=C, gamma=gamma, random_state=42)
    svm_classifier.fit(X_train, y_train)
    y_pred = svm_classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy with {kernel} : {accuracy * 100:.2f}%")
    return accuracy, y_pred


# Usage with different kernels:
kernels = ['linear', 'rbf', 'poly', 'sigmoid']
results = {}

for kernel in kernels:
    accuracy, y_pred = perform_svm(X_train, X_test, y_train, y_test, kernel=kernel)
    results[kernel] = {'accuracy': f'{accuracy * 100:.2f}%', 'predictions': y_pred}

best_kernel = max(results, key=lambda k: results[k]['accuracy'])
best_predictions = results[best_kernel]['predictions']
print(f"\nBest performing kernel: {best_kernel} with accuracy: {results[best_kernel]['accuracy']}")

# Evaluate the best model
print(f"\nClassification Report for {best_kernel}:")
print(classification_report(y_test, best_predictions))
```

```
Accuracy with linear : 71.68%
Accuracy with rbf : 97.05%
Accuracy with poly : 84.19%
Accuracy with sigmoid : 46.51%

Best performing kernel: rbf with accuracy: 97.05%

Classification Report for rbf:
              precision    recall  f1-score   support

           0       0.96      0.99      0.97      1330
           1       0.96      0.86      0.91       838
           2       0.97      0.99      0.98      4339
           3       0.98      0.96      0.97      1378

    accuracy                           0.97      7885
   macro avg       0.97      0.95      0.96      7885
weighted avg       0.97      0.97      0.97      7885
```

The **RBF kernel** as the **best-performing SVM** kernel with **97.05% accuracy**. The classification report highlights strong precision (0.96–0.98), recall (0.86–0.99), and f1-scores (0.91–0.97) across classes, despite class 1 having the lowest recall (0.86) due to imbalanced data.

Modelling and Evaluation

Hyperparameter Tuning

```python
# Hyperparameter Tuning using GridSearchCV
from sklearn.model_selection import GridSearchCV

param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [1, 0.1, 0.01, 0.001], 'kernel': [best_kernel]}
grid = GridSearchCV(SVC(random_state=42), param_grid, refit=True, verbose=3)
grid.fit(X_train, y_train)
print(f"\nBest parameters : {grid.best_params_}")
print(f"\nBest estimator : {grid.best_estimator_}")

grid_predictions = grid.predict(X_test)
print(f"\nClassification Report for {grid.best_params_} :")
print(classification_report(y_test, grid_predictions))
```

```
Best parameters : {'C': 100, 'gamma': 1, 'kernel': 'rbf'}

Best estimator : SVC(C=100, gamma=1, random_state=42)

Classification Report for {'C': 100, 'gamma': 1, 'kernel': 'rbf'} :
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1330
           1       0.98      0.99      0.98       838
           2       1.00      1.00      1.00      4339
           3       1.00      0.99      0.99      1378

    accuracy                           0.99      7885
   macro avg       0.99      0.99      0.99      7885
weighted avg       0.99      0.99      0.99      7885
```

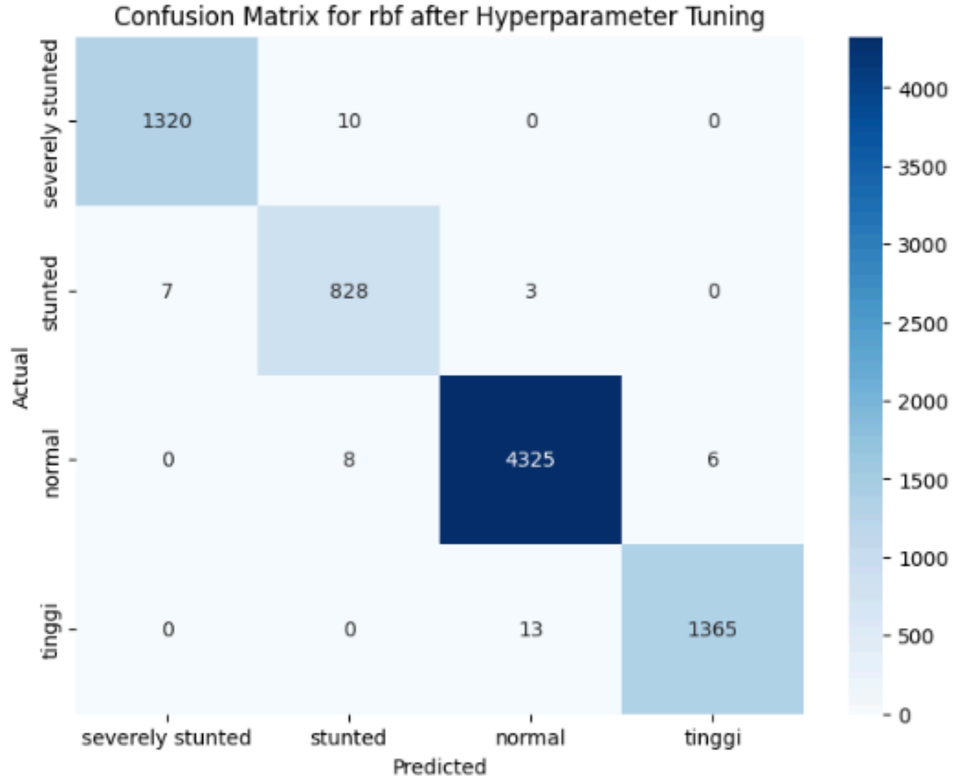
The output showcases hyperparameter tuning using GridSearchCV to optimize the SVM classifier's performance. The parameter grid includes values for C (0.1, 1, 10, 100), gamma (1, 0.1, 0.01, 0.001), and the kernel (set to 'rbf'). After fitting the model, the **best parameters** found are **C=100, gamma=1, and kernel='rbf'.**

Using these optimized parameters, the model achieves excellent performance with precision, recall, and f1-scores of 0.99 across all metrics. The accuracy is 99%, and both macro and weighted averages are also 0.99, indicating the model generalizes well across all classes. The hyperparameter tuning significantly improves the performance, yielding a highly effective model.

Modelling and Evaluation

Confusion Matrix

```python
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Confusion Matrix for the model after hyperparameter tuning
conf_matrix_grid = confusion_matrix(y_test, grid_predictions)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_grid, annot=True, fmt="d", cmap="Blues",
            xticklabels=['severely stunted', 'stunted', 'normal', 'tinggi'],
            yticklabels=['severely stunted', 'stunted', 'normal', 'tinggi'])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title(f"Confusion Matrix for {best_kernel} after Hyperparameter Tuning")
plt.show()
```



Conclusion

- The SVM model with optimized hyperparameters **(C=100, gamma=1, kernel='rbf')** delivers highly accurate results for stunting detection in toddlers.
- The model achieves **99% accuracy**, demonstrating its reliability in predicting stunting categories, including "severely stunted," "stunted," "normal," and "tall."
- Precision, recall, and f1-scores of 0.99 for all categories highlight the model's consistency and balance in its predictions, with minimal classification errors across all labels.

Appendix

SVM Classification Algorithm

Follow me !

www.linkedin.com/in/rahfaquraniyatin